Universidad de Salamanca

Escuela Politécnica Superior de Ávila

MÁSTER EN GEOTECNOLOGÍAS CARTOGRÁFICAS EN INGENIERÍA Y ARQUITECTURA

Trabajo Final de Máster:

Desarrollo de un complemento para QGIS para cargar los servicios cartográficos del Instituto Cartográfico Valenciano (ICV)

Autor:
Alfonso Moya Fuero

Directores:
Benjamín Arias Pérez; David Hernández López

Ávila – España
Septiembre 2020

ÍNDICE

Tabla de contenido

ÍNDICE DE FIGURAS.

1. Resumen.

Se implementa un complemento para QGIS que de forma automática, ofrezca los servicios cartográficos del visor de la Infraestructura de Datos Espaciales Valenciana (IDEV) mantenida por el Institut Cartogràfic Valencià (ICV) y puedan cargarse en el entorno de QGIS. El visor muestra un árbol de capas provenientes del catálogo estructurándolas por las temáticas especificadas en los anexos de la Directiva INSPIRE. Todos los servicios cargados tienen licencia Creative Commons 4.0.

Desde este árbol de capas se puede enlazar a los metadatos. En el panel del visor hay un servicio de búsqueda toponímica sobre el Nomenclàtor oficial, el complemento también usa este buscador para navegar por la cartografía. El portal de la IDEV tiene unos ficheros de configuración que sirven para mantener la estructura de las capas a mostrar de las diferentes Direcciones Generales de la Generalitat. Estos ficheros sirven de base para crear la estructura de árbol. El complemento sirve para tener toda la información de la IDEV integrada dentro del entorno de QGIS y la posibilidad de enlazarlo con la interfaz del catálogo para, por ejemplo, consultar el metadato o realizar la descarga de la cartografía.. Está disponible en GitHub y su código fuente es accesible por cualquier usuario.

Palabras clave: Cartografía base , IDEV, Nomenclátor, OGC, Complemento, QGIS

Abstract.

A plugin for QGIS is implemented that automatically offers the cartographic services of the Valencian Spatial Data Infrastructure (IDEV) viewer maintained by the Institut Cartogràfic Valencià (ICV) and can be loaded into the QGIS environment. The viewer shows a tree of layers from the catalog structuring them by the topics specified in the annexes of the INSPIRE Directive. All services are licensed under Creative Commons 4.0

From this layer tree you can link to the metadata. In the viewer panel there is a toponymic search service on the official Gazetteer, the plugin also uses this search engine to navigate the cartography. The IDEV portal has some configuration files that serve to maintain the structure of the layers to be displayed from the different General Directorates of the Generalitat. These files are the basis for creating the tree structure. The plugin is used to have all the IDEV information integrated within the QGIS environment and the possibility of linking it with the catalog interface to, for example, consult the metadata or download the cartography. It is available on GitHub and its source code is accessible by any user.

Keywords: Base mapping, IDEV, Gazetteer, OGC, Plugin, QGIS

2. Introducción.

En el año 2007 se publicó por parte del entones Consejo de Europa (actualmente Unión Europea) la Directiva 2007/2/CE (INSPIRE, 2007). Entre los objetivos de dicha directiva se encuentran la creación de una Infraestructura de Datos Espaciales (IDEs) común, basada en las creadas por los Estados miembros. Una IDE es «*Una Infraestructura de Datos de Espaciales (IDE) es un sistema informático integrado por un conjunto de recursos (catálogos, servidores, programas, aplicaciones, páginas web, ...) que permite el acceso y la gestión de conjuntos de datos y servicios geográficos (descritos a través de sus metadatos), disponibles en Internet, que cumple una serie normas, estándares y especificaciones que regulan y garantizan la interoperabilidad de la información geográfica*.». (Iniesto, M., & Núñez, A. ,2014)

Las directivas de la Unión Europea son de obligado cumplimiento, pero deben de transponerse en la legislación de cada estado miembro, dicha transposición al ordenamiento jurídico español, se realizó por medio de la Ley 14/2010, de 5 de julio, sobre las infraestructuras y los servicios de información geográfica en España (LISIGE, 2010), que también nos proporciona una definición formal de lo que es una infraestructura de datos espaciales, es una *«estructura virtual en red integrada por datos georreferenciados y servicios interoperables de información geográfica distribuidos en diferentes sistemas de información, accesible vía Internet con un mínimo de protocolos y especificaciones normalizadas que, además de los datos, sus descripciones mediante metadatos y los servicios interoperables de información geográfica, incluya las tecnologías de búsqueda y acceso a dichos datos; las normas para su producción, gestión y difusión; los acuerdos sobre su puesta en común, acceso y utilización entre sus productores y entre éstos y los usuarios; y los mecanismos, procesos y procedimientos de coordinación y seguimiento establecidos y gestionados de conformidad con lo dispuesto en la presente ley».*

Por lo tanto, existe un marco jurídico que obliga a cada estado miembro a implementar estas infraestructuras de datos, que, según esta directiva, deben de cumplir los siguiente cinco rasgos principales:

- Debe de ser posible combinar toda la información de forma continua y debe estar disponible de una forma no restrictiva.
- Debe de ser interoperable. Se define interoperabilidad como la habilidad de dos o más sistemas o componentes para intercambiar información y utilizar la información intercambiada (IEEE, 1990).
- Debe de contener servicios de descubrimiento y visualización gratuitos.
- Debe contener servicios de descubrimiento que informen de la disponibilidad, formato, ámbito, calidad y descripción de los datos.
- Los datos deben de estar disponibles en el formato más accesible e inteligible.

Por lo tanto, podemos resumir que el espíritu de la norma, (Capdevila, ,2004), se basa en que la información geográfica esté disponible a todos los niveles (desde el ciudadano, empresas hasta las administraciones), según unos estándares a nivel de normalización e interoperabilidad, evitando duplicidades, favoreciendo el mantenimiento de los datos por el organismo productor que mejor pueda hacerlo y promoviendo la cooperación y el intercambio.

Dentro del estado, la coordinación de todas las IDEs se realiza por el Consejo Superior Geográfico y la materialización del nodo central que coordina a todos es mediante el geoportal de la Infraestructura de Datos Espaciales de España, (IDEE ,2020), y que según indica en su apartado de introducción, *«tiene como objetivo integrar a través de Internet los datos, metadatos, servicios e información de tipo geográfico que se producen en España, a nivel estatal, autonómico y local, cumpliendo una serie de condiciones de interoperabilidad (normas, protocolos, especificaciones) y conforme a sus respectivos marcos legales. El fruto de este trabajo es el proyecto IDEE»*.

La arquitectura empleada para estas infraestructuras es la de cliente-servidor que se basa en un modelo en el que los clientes realizan peticiones (por ejemplo, el cliente, a través de un servicio WMS pide la imagen de un mapa para un encuadre determinado) y los servidores procesan dicha petición y devuelven el resultado (en este caso un conjunto de tiles o teselas en formato ráster que conforman la representación cartográfica del mapa). Por lo tanto necesitamos un cliente para acceder a los servicios de estas IDEs. El cliente puede ser de dos tipologías diferentes, ligero o pesado. Se define un cliente ligero como aquel que únicamente sirve de enlace entre los servidores (donde se realizan todas las tareas de procesamiento) y el usuario, su papel es de mero "intermediario". Por el contrario, un cliente pesado, realiza todas las tareas posibles en la computadora donde se ejecuta, repartiéndose las tareas de computación junto con los servidores.

Como ejemplo de cliente ligero tenemos los geoportales, que a través de una interfaz web, recoge las peticiones del cliente para mandarlas al servidor que las procesa y las devuelve. Dentro de este tipo estaría el visor de la *IDEV*. Como ejemplo de cliente pesado tenemos software de escritorio, que tiene capacidades propias de procesamiento. Dentro de este tipo estaría el software de sistemas de información geográfica *QGIS.*

Dentro de las IDEs se encuentran (implementados con diferentes tecnologías) los servicios de catálogo o descubrimiento (CSW), de visualización (WMS, WMS-C y WMTS) y de descarga (WFS o Atom). Además, todos ofrecen servicios de búsqueda toponímica sobre el Nomenclátor oficial. Mayoritariamente, es a través del visor del geoportal (catalogado como cliente ligero), desde donde los usuarios consumen los referidos servicios ya que suelen estar integrados todos ellos en la misma interfaz o API (Application Programming Interface). Desde este visor podemos visualizar las diferentes capas de cartografía (cargando en las diferentes APIs cartográficas las capas, que se muestran a su vez en estructura de árbol), realizar búsquedas toponímicas para navegar por la cartografía, y una vez posicionados en un ámbito determinado, y con las capas visualizadas, consultar los metadatos de estas o enlazar a su descarga. Las normas que regulan la implementación de los servicios son las siguientes:

- Servicios de visualización: ISO 19128:2005.
- Servicio de catálogo: ISO 19115:2003/ISO 19119:2015.
- Servicio de descarga: ISO 19142:2010

Por otro lado, tenemos los software de Sistemas de Información Geográfica como QGIS (catalogado como cliente pesado), que son capaces de cargar todos los servicios ofrecidos por las diferentes IDEs ya que estos están normalizados y cumplen con los estándares definidos por la ISO/TC211 (2002), y por el Open Geospatial Consortium, OGC (2020). Pero que los SIG posean estas capacidades, no basta para que el usuario escape de la consulta directa en los catálogos de datos para poder consumir los servicios

cartográficos, es necesario la programación de un complemento que haga de puente entre las IDEs y los SIG y que evite al usuario crear las conexiones y facilite la carga de las diferentes capas en sus proyectos de SIG.

Por tanto, las IDEs como los softwares tienden a compartir recursos, produciéndose una evolución de los SIG hacia las IDE de tal forma, que integran dentro de estos, los recursos ofrecidos por estas infraestructuras. Esa integración no es únicamente porque son capaces de consumir dichas herramientas, sino que cada vez es más común que tengan desarrollos propios para soportar y personalizar el acceso a dichos repositorios, aumentando la capacidad de los SIG al integrar esta ingente cantidad de datos (Rodríguez, 2008).

Al integrar dentro de los SIG los servicios de las IDEs, de una forma amigable y sencilla, se consigue aunar en el mismo entorno de trabajo, las capacidades de consulta y visualización de los geoportales desarrollados en las diferentes infraestructura de datos, junto con las capacidades de análisis y procesamiento de los SIG, facilitando las tareas, por ejemplo, de los técnicos que tienen que consumir los datos oficiales de las administraciones para la realización de proyectos con las herramientas SIG. Un campo donde se ve clara esta sinergia es en el Urbanismo y la Ordenación del Territorio, en la que los Planes Generales de Ordenación tienen que proyectarse sobre las cartografías oficiales (series cartográficas, catastro, planeamiento, lugares protegidos, infraestructura viaria, cartografías de riesgos,…) y a su vez, tienen que realizar análisis para la toma de decisiones en la correcta delineación de la calificación y clasificación del suelo, o el correcto establecimiento de los sectores de gestión, por lo tanto, esta integración de las IDEs en los SIG son una herramienta muy útil y justifica plenamente su implementación.

En consecuencia, se ha implementado a través de un complemento de QGIS las funcionalidades de un cliente ligero como es el visor de la IDEV (panel de gestión de capas, buscador toponímico y enlace con el catálogo de metadatos), dentro de un cliente pesado, como es QGIS, donde se pueden explotar conjuntamente todas las funcionalidades de la infraestructura de datos espaciales y las capacidades de geoprocesamiento de dicho software..

3. Estado del arte.

En la actualidad, existen diversos complementos para QGIS que realizan tareas similares a las que se ofrecen desde el geoportal de una IDE, ya que consumen de forma integrada en el software los servicios más habituales, entre los que podemos citar la búsqueda toponímica personalizada o la carga de los servicios cartográficos de la organización. Se he realizado una búsqueda de dichos complementos a partir de la funcionalidad de QGIS de "*Administrar e instalar complementos*" accesible desde el menú "*Complementos*". Se ha intentado identificar a las principales IDEs nacionales para verificar si disponen de alguna funcionalidad similar a la que se quiere desarrollar como objeto de este trabajo final de master. Se ha verificado la poca cantidad de estas herramientas desarrolladas para QGIS a pesar de su utilidad. También se han buscado para IDEs internacionales o complementos que, sin pertenecer a un organismo determinado, realizan las funciones que se quieren implementar (como es el caso del complemento *"QuickMapServices"* que aúna servicios cartográficos gratuitos en la red). Este es el listado de los complementos estudiados:

- IDE de Canarias (2020)
- QuickMapServices (2020)
- Spanish Inspire Catastral Downloader (2020)
- MTOPOOpenData (2020).
- Canadian Web Services (2020).
- Geoportal IDESC (2020).
- French Addres (2020).

A continuación, se citan algunos junto con una descripción de sus funcionalidades.

- IDE de Canarias (2020).

Ilustración 1. - Descripción del complemento IDECanarias.

Este complemento permite la búsqueda toponímica sobre las Islas Canarias. Una vez invocado desde el botón de la barra de herramientas o mediante el menú "*Complementos / Búsquedas IDE Canarias*", se presenta un panel en el que hay que introducir en el cuadro de texto, el patrón toponímico de búsqueda. Al pulsar "*Enter*", se presenta debajo a modo de lista, las entradas ordenadas de mayor a menor coincidencia.

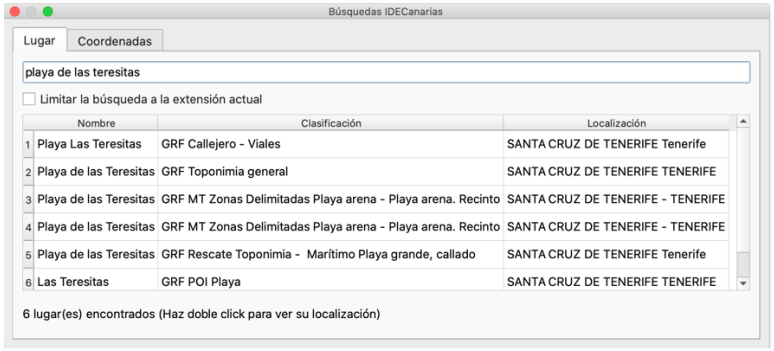


Ilustración 2. - Resultado de la búsqueda toponímica del complemento "IDECanarias".

Una vez hemos seleccionada la entrada deseada, por medio de "*doble click*" encima de ella, se carga la geometría del topónimo y centra la vista del mapa en el encuadre de dicha geometría.

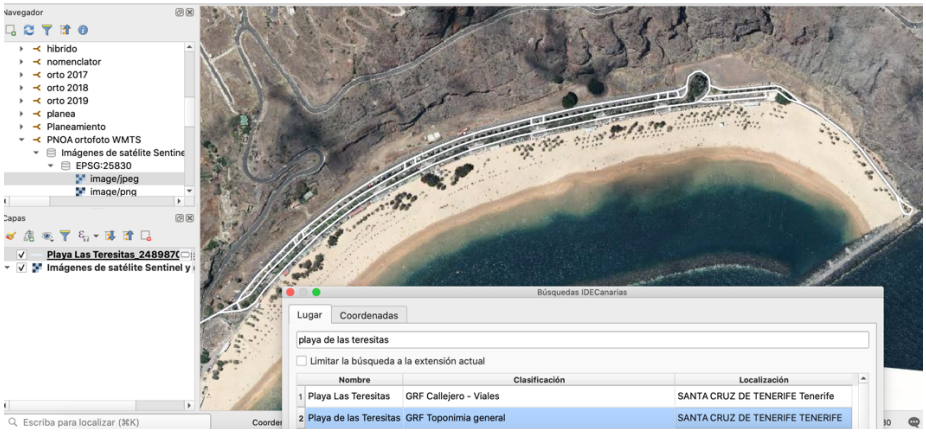


Ilustración 3. - Carga de la geometría de la búsqueda toponímica.

Se carga la geometría, en este caso correspondiente a "*Playa de Las Teresitas*", en una capa temporal o "*in_memory*". Cuando se carga, hace la transformación al SRC correspondiente al proyecto. Como utilidad complementaria tiene una segunda pestaña el panel, llamado "*Coordenadas*" que sirve para transformarlas en la proyección UTM (metros) o en el sistema global de coordenadas geodésicas (grados) y la utilidad de "*Ir a coordenadas*". También se puede usar el complemento "*Instituto Canario de Estadística (ISTAC)*" que sirve para la descarga de datos en formato de tablas en base a unos indicadores ya preconfigurados, o se puede enlazar con servicios WMS de visualización de dicha IDE. Al pulsar sobre "Obtener datos" u "Obtener cartografía" se cargan las capas en QGIS.

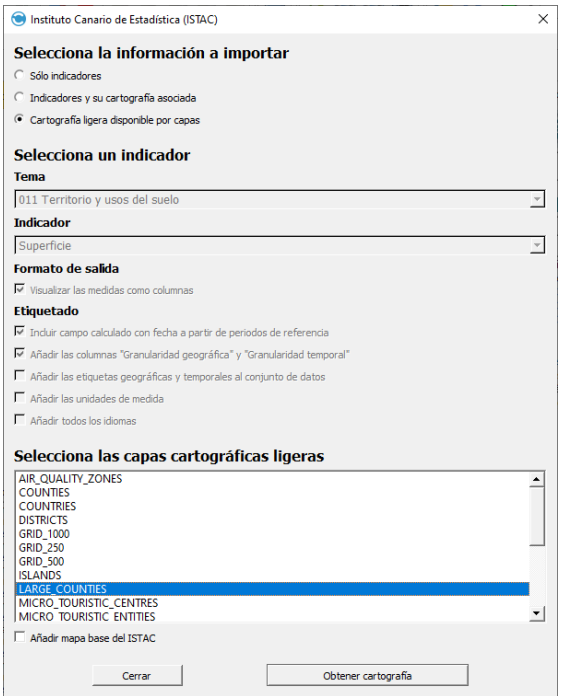


Ilustración 4. - Carga de los servicios WMS en el proyecto.

.

La funcionalidad de los complementos es la carga de datos alfanuméricos de los distintos indicadores del Instituto Canario de Estadística, así como los servicios WMS de la IDE de Canarias. Se complementa con un buscador toponímico que carga la cartografía del Nomenclátor canario.

- QuickMapServices (2020). "*Collection of easy to add basemaps*".

Ilustración 5. - Descripción del complemento "QuickMapServices".

Este complemento permite la carga a través del menú "*Web / QuickMapServices*", de numerosas capas base de diferentes proveedores. Al instalar el complemento, tiene un numero pequeño de servicios disponibles para cargar (los más comunes de la red como OpenStreetMap, Bing o Google), que se pueden ampliar entrando a la configuración del complemento a través del menú correspondiente, y descargando de la red una lista completa de servicios. Como nota negativa, algunos de los servicios enlazados no funcionan o están obsoletos.

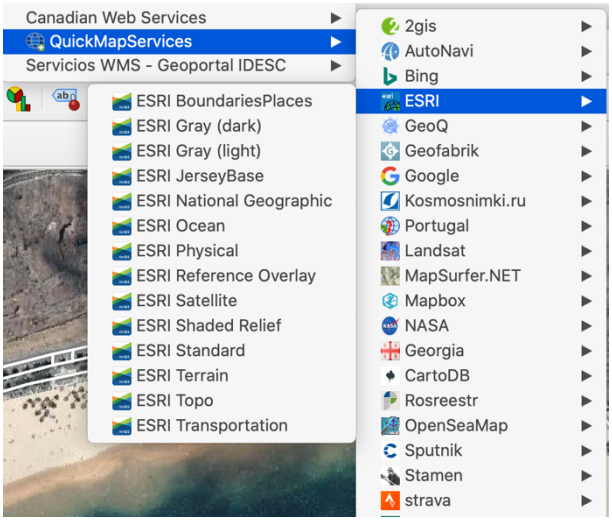


Ilustración 6. - Listado de servicios accesibles del complemento "QuickMapServices".

Una vez seleccionada el servicio base deseado, se carga en la tabla de contenidos como una capa más, pudiendo trabajar con ella al igual que con el resto. Esta utilidad ofrece un número muy alto de capas base, mayoritariamente en servicios WMTS y XYZ de teselas (servicios ráster cacheados que se cargan muy rápidamente en el mapa).

Otra utilidad es la búsqueda de los servicios en el encuadre del proyecto, mediante un patrón de texto, por lo que ofrece todas las capas base disponibles en ese *extent (*ámbito geográfico de la ventana del proyecto, también denominado como *BoundingBox)* y que tienen en su descripción el patrón de búsqueda. Esto evita cargar servicios que no tienen datos en el encuadre actual del proyecto. Se accede a esa utilidad desde el menú "*Web / QuickMapServices / Search QSM*". Indica en el propio panel de resultados, si ese servicio está operativo o no, mediante un puntito verde o rojo. Pulsando en el botón "Add" se produce la carga.

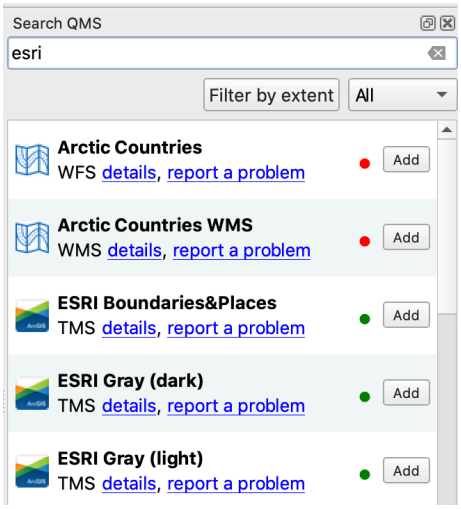


Ilustración 7. - Resultado de la utilidad de búsqueda de servicios en el encuadre actual.

La principal funcionalidad es la carga de servicios cartográficos estándar de diferentes proveedores, junto con una búsqueda de servicios basado en el encuadre geográfico del proyecto de QGIS, siendo esta funcionalidad de gran utilidad ya que evita la carga de servicios que no cubren el espacio de trabajo.

- Spanish Inspire Catastral Downloader (2020).

Ilustración 8. - Descripción del complemento" Spanish Inspire Catastral Downloader".

Este complemento permite la descarga de la cartografía oficial catastral para INSPIRE, en formato GML, y una vez descargada permite su carga en la tabla de contenidos. Hace uso de unos servicios *Atom* de descarga ofrecidos por la sede virtual de Catastro disponibles en la url http://www.catastro.minhap.gob.es/webinspire/index.html Se muestra, por medio de desplegables, la posibilidad de seleccionar el municipio requerido (previa selección de la provincia). Una vez escogido el ámbito geográfico, hay que especificar el directorio de descarga (donde se descomprimirán los datos que originalmente están en .zip) y las capas a descargar (hay tres posibilidades, parcelario, edificios o direcciones). Estos servicios *Atom* se basan en un listado de descarga en formato *XML.* Para su visualización correcta se recomienda el navegador "*Internet Explorar",* con "*Firefox*" o "*Google Chrome*" no se representan bien formateados.

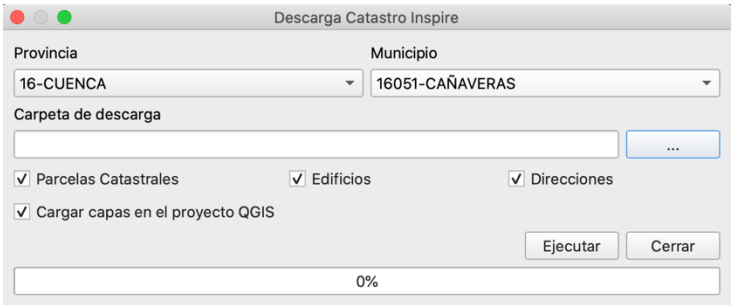


Ilustración 9. - Diálogo de descarga del complemento "Spanish Catastral for INSPIRE download".

Una vez descarga la cartografía se carga en el proyecto, pudiendo trabajar con ella como una capa más de la tabla de contenidos. La principal funcionalidad es la descarga de las capas catastrales adaptadas a la especificación Inspire para dicho conjunto de datos, en formato GML.

- MTOPOOpenData (2020).

Este complemento sirve para acceder a los servicios de la Infraestructura de Datos del Ministerio de Obras públicas de Uruguay. Se basa en un complemento previo denominado "*GreekOpenData*". Se invoca desde el menú "*Complementos / MTOP Open Data / MTOPOpenData"* mostrándose un panel en el que la primera entrada del cuadro de texto sirve para filtrar por un patrón, las capas listadas en la búsqueda se muestran en la caja de abajo. Todas las capas se ordenan alfabéticamente por nombre e indican en las correspondientes columnas, el tipo de servicio (WMS o WFS) y la organización propietaria del dato.

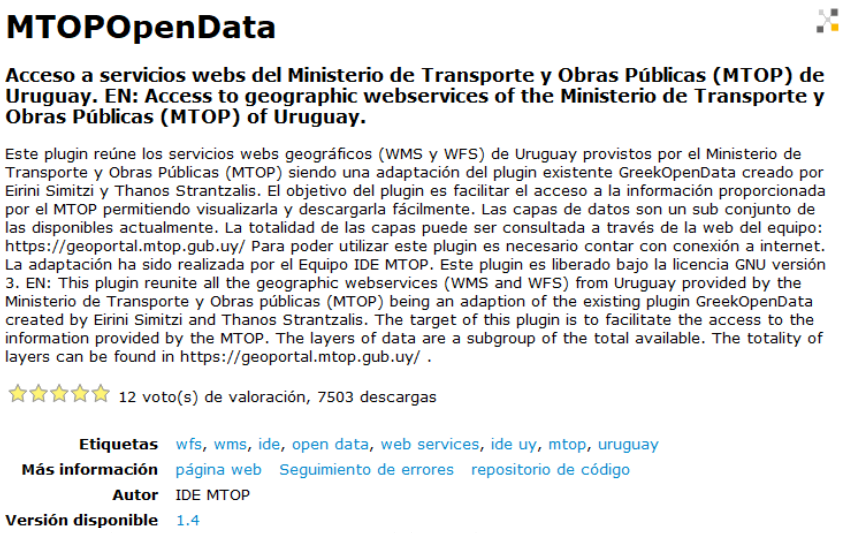


Ilustración 10. - Descripción del complemento "MTOPOpenData".

En la parte de abajo se muestra información de la capa, como una descripción, fecha de la última actualización cartográfica y una previsualización del conjunto de datos representado. También se puede pulsar el botón de "*Info*" que enlaza con el catálogo del geoportal de Uruguay mostrando la información publica del dato.

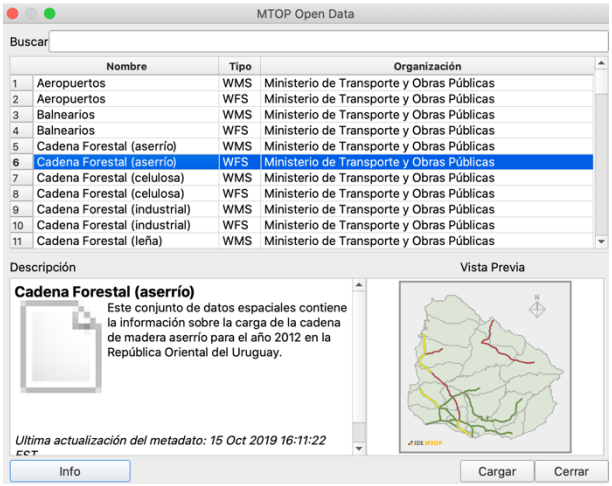


Ilustración 11. - Selección del servicio a cargar del complemento "MTOPOpenData".

El complemento ofrece un total de 104 capas. La principal funcionalidad es la carga de servicios cartográficos con el enlace al catálogo de su IDE correspondiente.

- Canadian Web Services (2020).

Este complemento es similar al anterior, muestra un total de 904 capas. En la lista de servicios, ofrece el nombre del servicio, el tipo (WMS, WFS o ESRI Webservice), el host y el número de capas que contiene. Una vez seleccionado muestra en un recuadro inferior, información de la capa correspondiente. A través del botón "*Info*" indica el tipo de capas que sirve y el origen.

Ilustración 12. - Descripción del complemento "Canadian Web Services".

Una vez tenemos seleccionado el servicio, por medio del botón "*Load*" cargamos las capas en la tabla de contenidos. La principal función de este complemento es la búsqueda por texto de los servicios cartográficos y la carga de los mismos a partir del listado resultante de la búsqueda, en el que se indica el proveedor, así como el tipo de estándar utilizado. También ofrece la posibilidad de ordenar el listado de servicios mostrado en la tabla, en un despegable se indica el campo en el cual basar la ordenación alfabética.

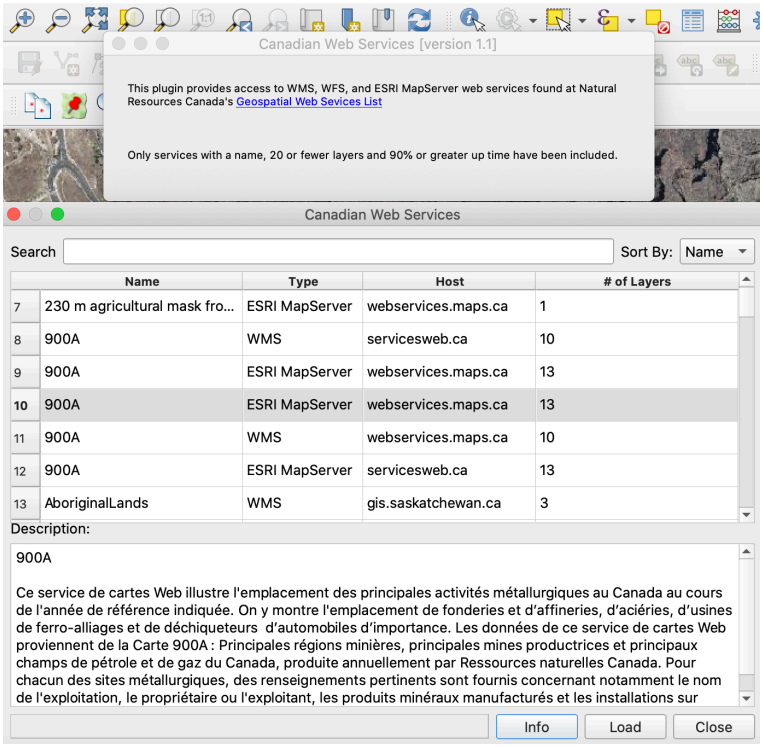


Ilustración 13.- Carga de servicios del complemento de "Canadian Web Services".

Como nota positiva, este complemento solo carga aquellas capas que se encuentran actualizadas y es capaz de cargar servicios de diferentes tecnologías aunque no sean un estándar, como son los servicios de Arcgis Server. Otro punto a favor es que no es necesario cargar la capa para obtener los metadatos de la misma ya que aparecen en el mismo diálogo de carga. Sin embargo, el nombre de algunas capas es poco descriptivo teniendo por lo que dificulta la búsqueda por nombre de capa, hay que acudir a su descripción para averiguar la temática del servicio.

- Geoportal IDESC (2020).

Ilustración 14. - Descripción del complemento de Servicios WMS – IDES Cali.

Este complemento es similar al anterior, están los dos basados en el mismo código fuente, aprovechando el código fuente compartido por los complementos y la licencia tipo "*Creative Commons 4.*0" de los mismos. Tiene las mismas funcionalidades que el anterior y utiliza los mismos botones e interfaz para mostrar las capas a cargar.

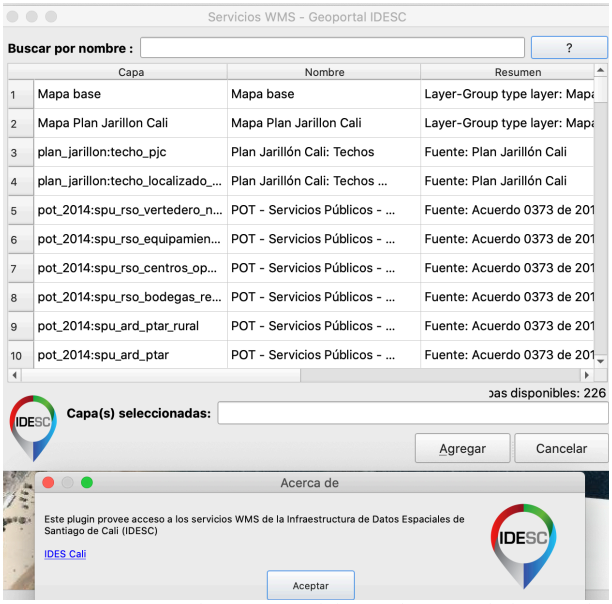


Ilustración 15. - Carga de servicios del complemento de la IDE de Cali.

La funcionalidad principal, al igual que en complemento anterior, es la búsqueda de servicios cartográficos a partir de texto y su carga en el proyecto.

- French Addres (2020).

Este complemento es del tipo de buscador toponímico, basado en las direcciones oficiales de la cartografía de Francia. El gobierno francés pone a disposición de la ciudadanía un servicio de geocodificación (directa e inversa) a través de una API, en la dirección https://geo.api.gouv.fr/adresse El servicio se invoca por una petición HTTPS (funciona con los métodos GET y POST) mediante un patrón de búsqueda textual y devuelve un fichero en formato GeoJSON.

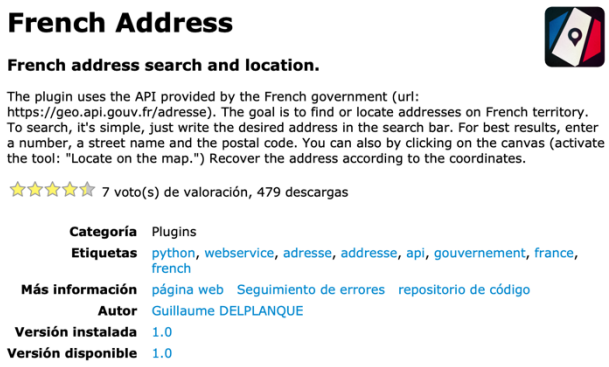


Ilustración 16. - Descripción del complemento "French Address".

Se invoca a través del botón correspondiente de la barra de herramientas o a través del menú "*Complementos / Adresse France*". Los servicios de geocodificación están descritos en el geoportal de la IDE del "*Institut Géographique National*" de Francia.

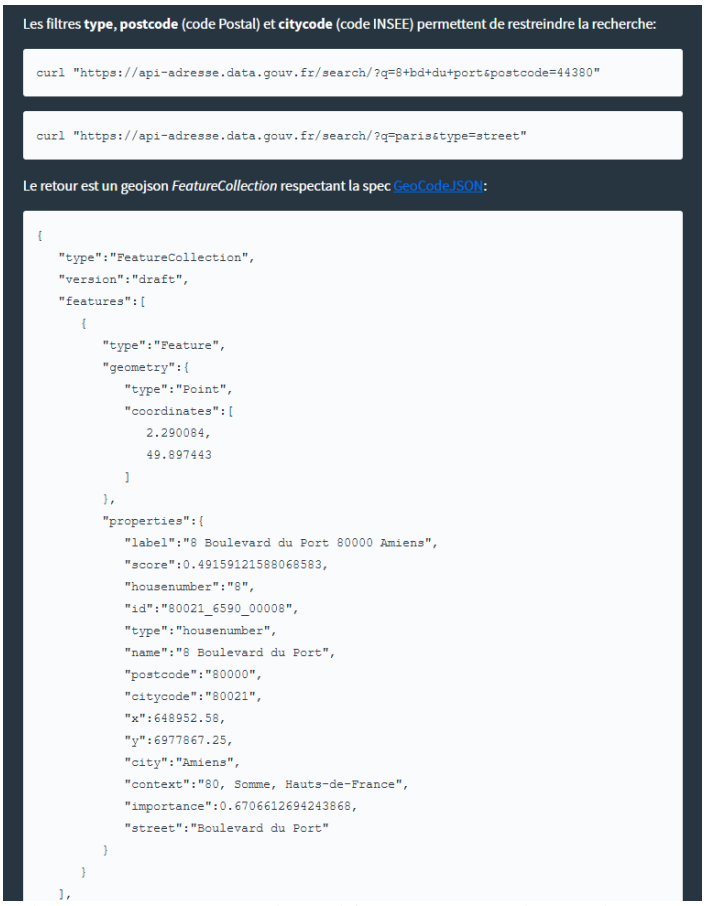


Ilustración 17. - Servicio de geodificación a través de un webservice.

Una vez invocado, se abre un panel lateral en la parte izquierda que sirve para introducir un patrón de búsqueda, centrando el encuadre en la dirección postal indicada. La georreferenciación puede ser directa (a partirde una dirección postal) o inversa (a partir de unas coordenadas cartográficas).

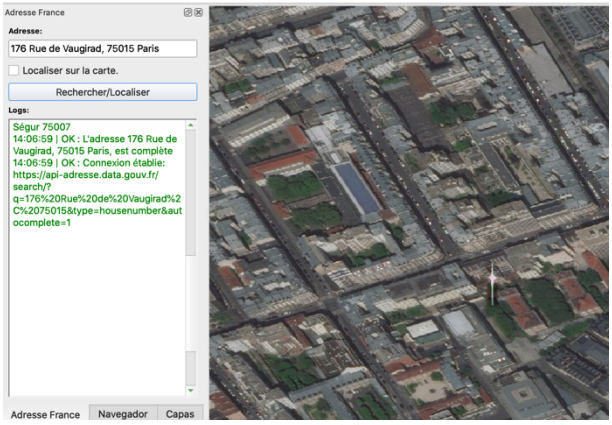
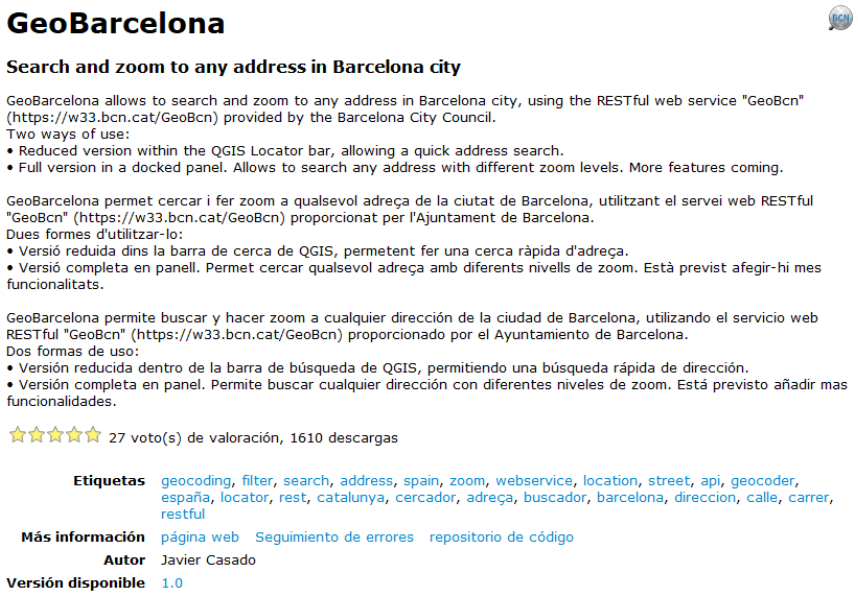


Ilustración 18. - Buscador toponímico del complemento French Address.

La principal funcionalidad es el nomenclátor, implementado sobre una API oficial que devuelve un fichero GeoJSON con la información de la geocodificación. El complemento encuadra automáticamente a la localización encontrada mediante doble clic sobre el resultado deseado. De igual manera, ofrece en un cuadro de texto toda la información referente a la conexión con el servicio de georreferenciación.

- GeoBarcelona (2020).

Este complemento ofrece búsquedas toponímicas de direcciones postales sobre la ciudad de Barcelona utilizando el servicio web "*GeoBcn*" a través de la url *https://w33.bcn.cat/GeoBcn* proporcionado por el Ayuntamiento desde su IDE.



GeoBarcelona

Search and zoom to any address in Barcelona city

GeoBarcelona allows to search and zoom to any address in Barcelona city, using the RESTful web service "GeoBcn" (https://w33.bcn.cat/GeoBcn) provided by the Barcelona City Council.
Two ways of use:
• Reduced version within the QGIS Locator bar, allowing a quick address search.
• Full version in a docked panel. Allows to search any address with different zoom levels. More features coming.

GeoBarcelona permet cercar i fer zoom a qualsevol adreça de la ciutat de Barcelona, utilitzant el servei web RESTful "GeoBcn" (https://w33.bcn.cat/GeoBcn) proporcionat per l'Ajuntament de Barcelona.
Dues formes d'utilitzar-lo:
• Versió reduida dins la barra de cerca de QGIS, permetent fer una cerca ràpida d'adreça.
• Versió completa en panell. Permet cercar qualsevol adreça amb diferents nivells de zoom. Està previst afegir-hi mes funcionalitats.

GeoBarcelona permite buscar y hacer zoom a cualquier dirección de la ciudad de Barcelona, utilizando el servicio web RESTful "GeoBcn" (https://w33.bcn.cat/GeoBcn) proporcionado por el Ayuntamiento de Barcelona.
Dos formas de uso:
• Versión reducida dentro de la barra de búsqueda de QGIS, permitiendo una búsqueda rápida de dirección.
• Versión completa en panel. Permite buscar cualquier dirección con diferentes niveles de zoom. Está previsto añadir mas funcionalidades.

27 voto(s) de valoración, 1610 descargas

Etiquetas geocoding, filter, search, address, spain, zoom, webservice, location, street, api, geocoder, españa, locator, rest, catalunya, cercador, adreça, buscador, barcelona, direccion, calle, carrer, restful

Más información página web Seguimiento de errores repositorio de código

Autor Javier Casado

Versión disponible 1.0

Ilustración 19. - Descripción del complemento "GeoBarcelona".

Se invoca a través del menú "*Complementos / GeoBarcelona / Abre-Cierra panel"* presentándose un diálogo con un cuadro de texto para introducir los patrones de texto para la búsqueda. Una vez seleccionado mediante doble clic, muestra un punto azul parpadeante durante 15 segundos, según consta en la ayuda.

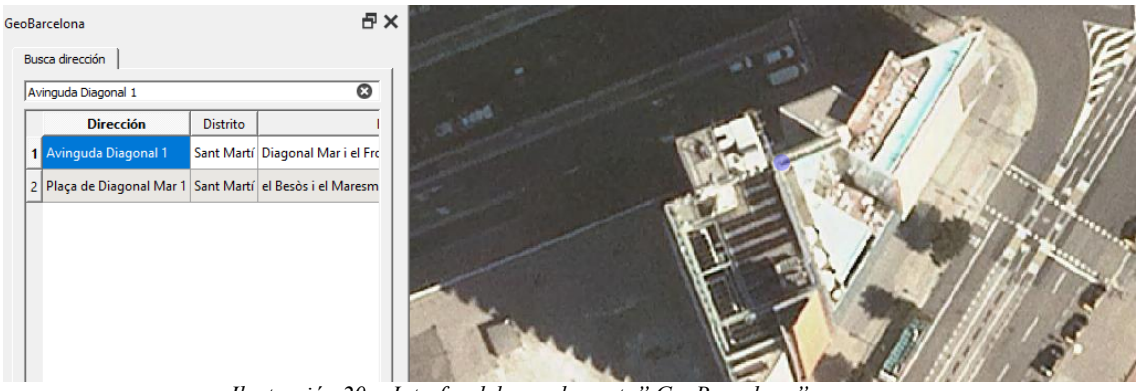


Ilustración 20. - Interfaz del complemento" GeoBarcelona".

La principal funcionalidad es la búsqueda toponímica en el nomenclátor del Ayuntamiento de Barcelona, centrando la búsqueda en el punto encontrado. Las direcciones se muestran en formato de tabla indicando el barrio y distrito al que pertenece.

4. Objetivos del trabajo.

Se pretende programar un complemento de QGIS que sea capaz de integrar dentro de dicho software, parte de las funcionalidades del visor de la IDEV. Hemos comprobado como la mayoría de los complementos relacionados con IDEs tienen dos funcionalidades recurrentes:

- Búsqueda toponímica.
- Gestión de capas de información.

Si entramos en el visor de la IDEV, apreciamos estas funcionalidades implementadas en la interfaz principal del visor:

- Buscador toponímico.
- Selector de capas base.
- Panel de capas.
- Enlace al catálogo de la IDEV (a través de la capa correspondiente).

El objetivo del trabajo es programar un complemento en QGIS desde el que se acceda a esas cuatro funcionalidades. Se realiza a través de un panel que se muestra dentro del entorno de trabajo. Este contenedor tiene que reproducir los más fielmente posible al visor de la IDEV, tanto la apariencia como el comportamiento de las capas que se gestionan mediante una estructura en árbol.

Si bien en el portal se ha programado con los leguajes *Html, CSS y JavaScript*, tecnologías que aumentan las posibilidades de apariencia y comportamiento de los controles en comparación con *PyQt* que en este sentido es más limitado, pero suficiente para el objetivo de replicar el gestor de capas de la IDEV. Se tiene que replicar el texto predictivo de las búsquedas (toponímica y de capas) y que el resultado se muestre en una lista desplegable. También se ha de replicar la estructura en árbol en que se estructuran todos los servicios cartográficos de la Generalitat y que se cargan / descargan mediante "*checkboxes*". Estas cuatro funcionalidades se invocan en los siguientes apartados y que se recogen la siguiente figura.

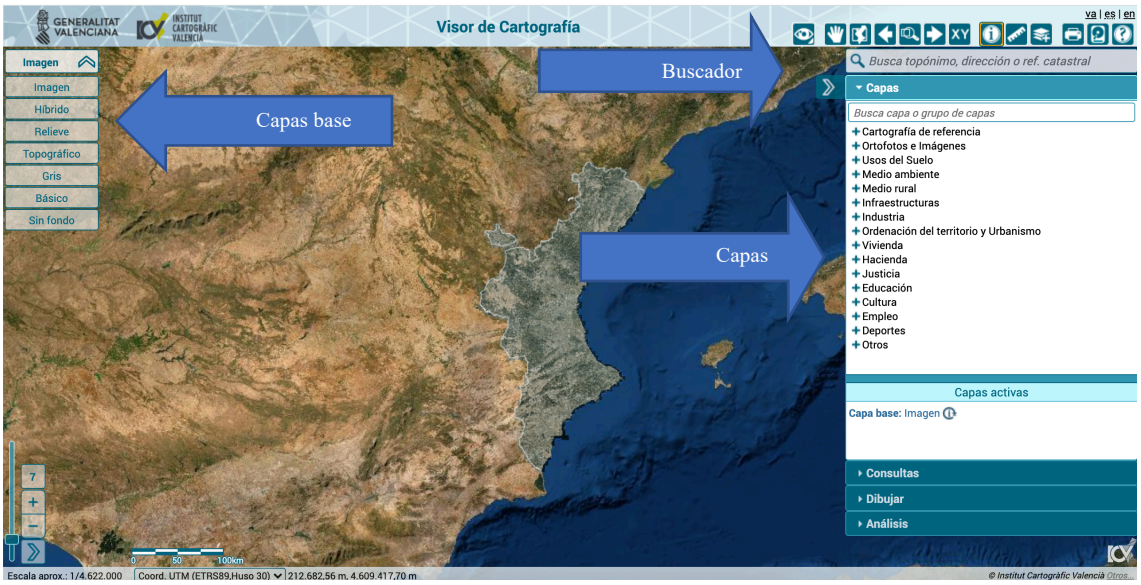


Ilustración 21. - Funcionalidades del visor de la IDEV.

5. Medios empleados.

Los medios empleados para la programación del complemento los podemos resumir en tres apartados:

- Plataforma de trabajo y entorno:

  Se utiliza como plataforma el software *QGIS* en su versión 3.10 de largo plazo, ya que el complemento se desarrolla para el funcionamiento dentro de la misma. El lenguaje de programación utilizado es *Python* en su versión 3.6.8.

- Plataforma de desarrollo:

  El complemento se programa en el lenguaje *Python,* utilizando las librerías de *PyQt (biblioteca gráfica de Python)* y *PyQGis* (biblioteca de *QGIS*), ambos son *bindings (*adaptaciones*)* de las versiones originales en *C++* de *Qt* y *API* de *QGIS.* Para la edición de los ficheros que componen el complemento se utiliza el entorno de desarrollo integrado *PyCharm* en su versión 2020.1.2 que utiliza el intérprete de *Python.* Se ha utilizado para la modificación de los ficheros en formato *PY (ejecutables de python)* y *JSON* (diccionario de datos). También se utiliza para el desarrollo los complementos "*Plugin Builder*" para crear la estructura inicial junto con los archivos necesarios para el correcto funcionamiento y el complemento "*Reload plugin*" que se encarga de recargar el complemento después de cada modificación para poder probarlo dentro del entorno de trabajo. Para la creación de la interfaz gráfica de usuario (panel flotante dentro de *QGIS*) se utiliza el programa *Qt Designer with QGIS 3.10.4* que se instala junto con la plataforma de trabajo y entorno. Dicho entorno accede directamente a los *widgets* (control visual o componente, por ejemplo, un botón) de *PyQt* y de *PyQGis*.

- Recursos disponibles:

  Se utilizan los recursos y servicios cartográficos de la *IDEV*. Para el buscador toponímico se utiliza el webservice del visor (implementado con la tecnología del indexador "*Solr" (2020),* que devuelve las coincidencias en formato *JSON*. Cada resultado tiene como atributos su descripción, tipo, encuadre y nombre oficial. Para los servicios de visualización se usan los servicios *WMS/WMTS* de la IDEV provenientes de las diferentes Direcciones Generales de la Generalitat, y agrupados en una estructura en árbol siguiendo exactamente la misma clasificación que en el visor de la IDEV. Para la consulta de los metadatos y acceso a los servicios de descarga se utiliza el Catálogo de la *IDEV*, cada capa enlaza con sus datos correspondientes a través del *UUID* (identificador único) de cada metadato. La información se muestra en el navegador usando la propia interfaz del catálogo. Para la ayuda se lanza una petición de apertura de un documento *HTML*, es el ordenador del cliente el que usando su navegador predeterminado el que lo muestra, fuera del entorno de *QGIS*.

6. Desarrollo del complemento.

En este apartado desarrollamos pormenorizadamente los pasos realizados hasta obtener el complemento totalmente operativo dentro del software QGIS.

1.1. Creación del complemento con “Plugin Builder”.

Usamos esta herramienta para crear el esqueleto del complemento. Al invocarlo, se nos presenta el siguiente diálogo en el que especificamos los parámetros principales como es el nombre del complemento, el nombre de la clase, datos de contacto y una descripción.

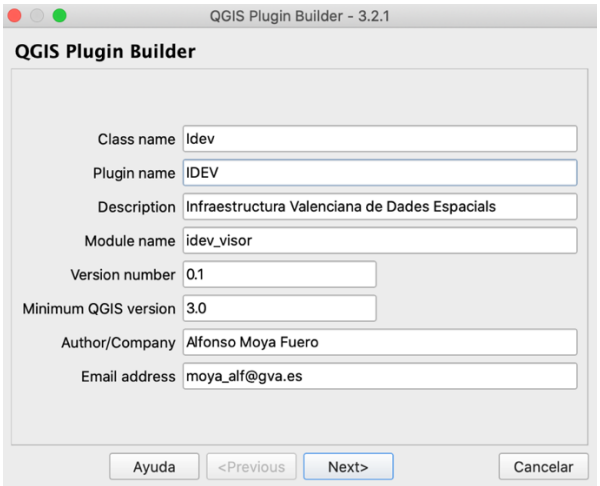


Ilustración 22 . - Creación del complemento con "Plugin Builder".

En el siguiente diálogo introducimos una pequeña descripción del funcionamiento del complemento y el objetivo del mismo. Es importante ir documentado correctamente estos apartados de los metadatos del complemento ya que en este texto se basa el “Buscador de complementos” que hay dentro de QGIS. Cuanto mas precisos seamos, mas probabilidades hay que el usuario que lo necesita, lo encuentre. Es recomendable incluir tanto una descripción funcional como del ámbito geográfico en que trabaja. Si lo realizamos en las lenguas oficiales de los potenciales usuarios también aumentaremos el número de descargas.

Ilustración 23 . - "Acerca de" del complemento.

La pantalla final sirve para configurar la apariencia del complemento, en este caso será un panel acoplable a la derecha y el nombre del menú para su invocación. En este caso elegimos "*Tool button with dock widget*" que nos crea el esqueleto de un panel que se invoca por medio de un botón en la barra de herramientas.

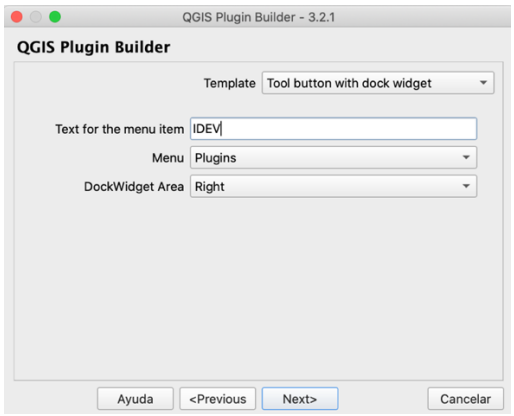


Ilustración 24 . - Configuración visual del complemento.

En el siguiente diálogo configuramos la manera de compilación y opciones generales.

Ilustración 25 .- Opciones del complemento.

Finalmente introducimos las direcciones del repositorio que hemos creado en GitHub para hospedar el código fuente y que sea accesible por la comunidad. También indicamos la web del portal de la IDEV como referencia de los servicios cargados en el complemento.

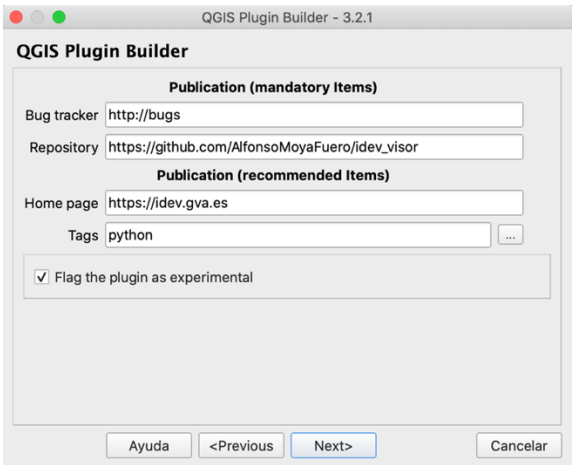


Ilustración 26 . - Direcciones de consulta del complemento.

Por defecto se crean todos los ficheros necesarios (el esqueleto del complemento) en el directorio del perfil del usuario por defecto de QGIS. Este directorio se puede consultar a través del menú "Configuración" / "Perfil de usuario" / "Abrir la carpeta del perfil activo". En la imagen se ve el directorio donde se crea, en este caso, se está desarrollando todo con un sistema operativo Mac OS.

Ilustración 27 . - Directorio de creación del complemento.

Al pulsar sobre el botón "Generate" se crea un complemento en blanco con las configuraciones especificadas. Estos ficheros son la base sobre los que se escribirá la programación. En nuestro caso únicamente se ha insertado líneas de código en el fichero "*idev_visor_dockwidget.py*". Este fichero contiene los componentes del panel que se invoca al iniciar el componente, por lo que definimos en el mismo, todas las funciones que se desencadenan cuando el usuario interactúa con los *widgets* (como por ejemplo, al pulsar un botón o activar el checkbox de una capa, se recoge el evento y se invoca un método que realiza una determinada tarea, como es abrir una ayuda o cargar una capa en *QGIS*).

Después de generarlo QGIS nos hace la advertencia mostrada en la siguiente figura, el complemento se ha creado, pero no se ha compilado (hemos desactivado esa opción al crearlo). Se tendrá que compilar utilizando la consola del sistema.

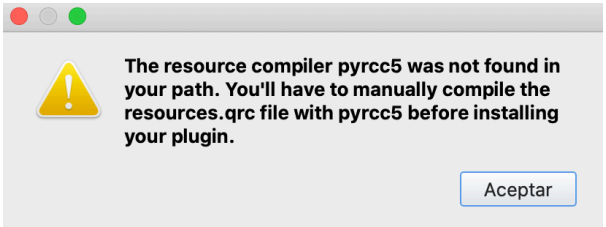


Ilustración 28 . - Advertencia de compilación al crear el complemento.

Después se nos presenta la siguiente ventana en la que nos indica los pasos a seguir para la compilación, nos indica como cambiar el icono de invocación que aparecerá en la barra de herramienta, nos informa del fichero que contiene la interfaz de usuario y donde podemos consultar ayuda para el desarrollo de complementos.

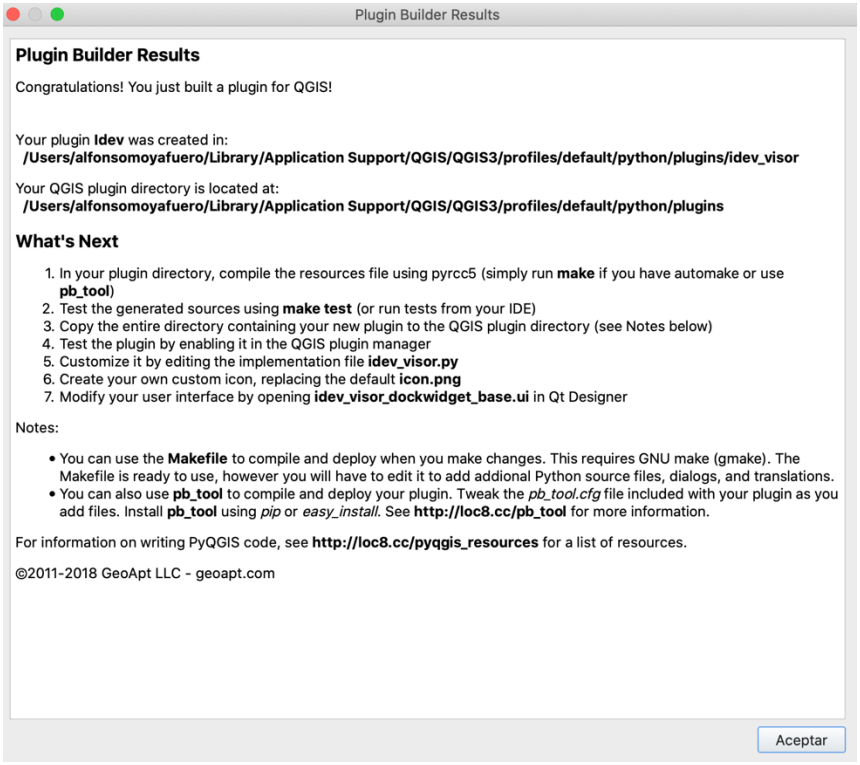


Ilustración 29 . - Ventana resumen de la creación del complemento,

En la siguiente figura vemos los ficheros que se han generado entre los que destacamos:

- *idev_visor_dockwidget.py* .- Código fuente del panel, en el será donde insertemos todas las líneas de código ya que es en ese panel en el que añadiremos todos los controles de la interfaz.
- *params.json* .- Fichero en formato json que se usa par acerar el diccionario de capas dentro del complemento. en el que se indica el nombre de capa, avisos, metadato y url de conexión.
- *idev_visor_dockwidget_base.ui* .- Fichero de configuración de la interfaz, en este caso es el fichero que editaremos con el programa "Qt Designer for QGIS"
- *_init_.py* .- Fichero en el que se define la clase "iface", son todos los controles propios de QGIS y que utilizaremos para controlar las acciones del programa como por ejemplo, cargar capas o cambiar zoom.
- *idev_visor.py* .- Constructor principal del complemento, en este fichero estarían definidas todas las ventanas que se utilizan, en este caso, solo tenemos un panel principal.
- *metadata.txt* .- Textos de la información que ofrece el complemento en el gestor de complementos de QGIS e información acerca del mismo.
- *resources.qrc* .- Fichero que contiene referencias a otros recursos utilizados, como por ejemplo, el fichero de icono en formato .png .
- LICENSE.html . - Términos de licencia del complemento, que está bajo *Creative Commons 4.0* El fichero describe los términos de dicha licencia, se permite copiar y redistribuir el material en cualquier medio o formato, adaptarlo, extenderlo, modificarlo incluso comercialmente, siempre que se cite la fuente original del mismo y se expliquen los cambios aportados.

1.2. Creación de la interfaz mediante "Qt Designer for QGIS".

Para la creación de la interfaz del complemento, el propio QGIS tiene la herramienta *Qt Dessigner with QGIS 3.10.5,* esta es una herramienta desarrollada para el uso de las librerías de *Qt* y que resulta muy útil cuando se programa en *Python* porque integra las librerías PyQt (utilizadas en este complemento. Permite crear el fichero "*idev_visor_docketwidget_base.ui"* que describe la interfaz de usuario. Desde la interfaz del programa, de una forma visual, se van arrastrando sobre el panel del complemento, todos los *widgets* empleados y que se listan a continuación:

- QLabel .- Para los textos del panel e imágenes.
- QPushButton .- Botones para abrir las webs de la IDEV o consulta del catálogo para la capa seleccionada en el árbol de capas.
- QLineEdit .- Contenedor de listas de texto, utilizado para mostrar los resultados de la búsqueda toponímica o la búsqueda de capas.
- QRadioButton .- *Widgets* para cambiar el idioma del complemento, siempre hay uno de los dos activados, por defecto el complemento se carga en castellano.
- QTreeWidget .- *Widget* que crea una estructura de árbol y que se utiliza para ordenar todas las capas que se cargan de la IDEV. La estructura de los nodos (padres e hijos) es la misma que la utilizada en el visor de la IDEV para que al usuario le sea familiar y sea capaz de encontrar sin dificultad el servicio requerido.

Desde la aplicación se han ido añadiendo cada uno de los componentes utilizados, todos tienen un texto de ayuda o "tooltip" que cambia según el idioma elegido. De igual manera, se invocan desde el código fuente para cambiar sus propiedades y lanzar las funciones correspondientes cuando un evento se produce en el panel, por ejemplo, cuando se activa una capa en el árbol, se carga la correspondiente capa en el proyecto, o cuando se escribe texto para la búsqueda toponímica, se presentan las coincidencias en una lista. En la siguiente figura se muestra el panel configurado dentro de la herramienta Qt Designer (esta herramienta únicamente está en la distribución de QGIS para Windows, se ha tenido que recurrir a una máquina virtual para poder utilizarla):

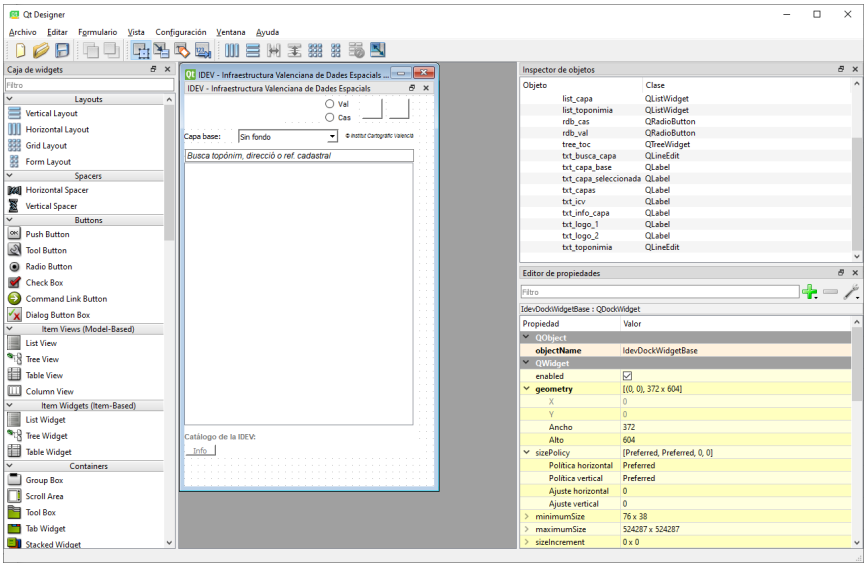


Ilustración 30. - Interfaz del programa "Qt Designer with QGIS 3.10.5".

Los iconos e imágenes utilizados en el complemento se cargan por código por lo que no aparecen en esta configuración. La interfaz crea un fichero con la extensión .ui (es el acrónimo de "*user interface"*) y tiene una estructura de lenguaje de marcas (XML). Este fichero se puede editar directamente con un editor de texto, pero es más recomendable hacer todos los cambios a través de la herramienta antes expuesta para no cometer fallos de sintaxis que pueden corromper la interfaz.

```
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
 <class>IdevDockWidgetBase</class>
 <widget class="QDockWidget" name="IdevDockWidgetBase">
  <property name="geometry">
   <rect>
    <x>0</x>
    <y>0</y>
    <width>372</width>
    <height>604</height>
   </rect>
  </property>
  <property name="styleSheet">
   <string notr="true">background-color: rgb(255, 255, 255);</string>
  </property>
  <property name="windowTitle">
   <string>IDEV - Infraestructura Valenciana de Dades Espacials</string>
  </property>
  <widget class="QWidget" name="dockWidgetContents">
   <widget class="QLabel" name="txt_logo_1">
    <property name="enabled">
     <bool>false</bool>
    </property>
    <property name="geometry">
     <rect>
      <x>0</x>
      <y>0</y>
      <width>145</width>
      <height>48</height>
     </rect>
    </property>
```

Ilustración 31. - Fichero .ui que contiene los widgets usados en el panel del complemento.

Una vez se carga en QGIS el complemento a través del menú "*Complementos*" / "*Administrar e instalar complementos*", aparece el botón de invocación en la barra de herramientas.

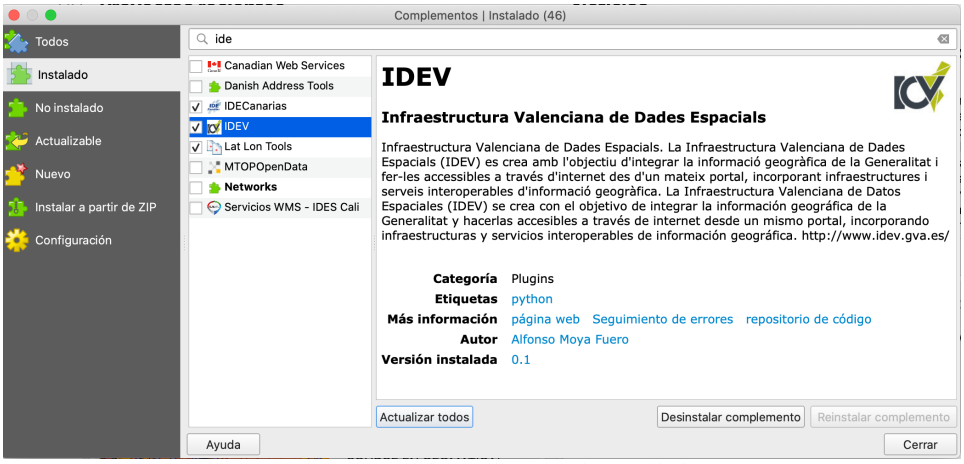


Ilustración 32. - Carga del complemento desarrollado en QIGS.

Además del botón de invocación, también se crea una entrada en el menú "*Complementos*" / "*IDEV*" / "*IDEV*", este sería su aspecto visual:

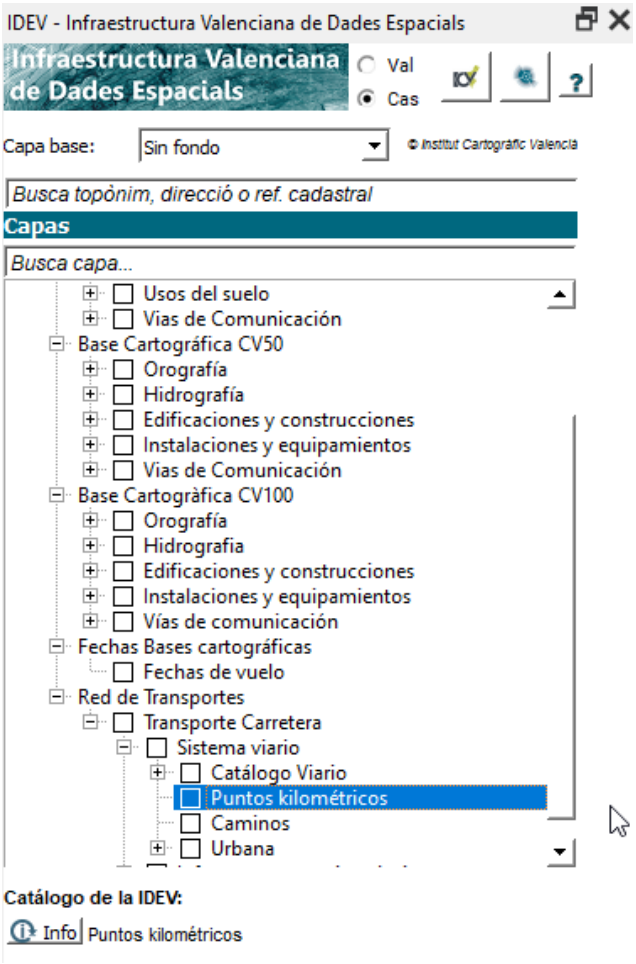


Ilustración 33. - Panel del complemento dentro de QGIS.

1.3. Programación del complemento.

Para la programación en Python del complemento se ha utilizado la herramienta *PyCharm* que es una *IDE (integrated development environment),* es decir, un entorno completo de desarrollo que contiene todas las librerias utilizadas y que permite invocar todas las propiedades y funciones asociadas a los *widgets* empleados mediante asistentes a la escritura, por ejemplo, cuando se invoca un *widget* tipo QLabel, al pulsar el carácter "." se despliegan automáticamente todas las propiedades o funciones de este widget pudiendo invocarlas con los cursores, evitando así cometer errores sintácticos y de paso, sirve de ayuda para ver todos los métodos que contiene, sin necesidad de acudir a la ayuda oficial para obtener este listado. También incluye herramientas de refactorización, formateo e identado del código fuente (recordemos que, en Python, se usan los tabuladores para jerarquizar el código, de esta manera se marca si una línea está dentro de una rutina o no, o pertenece a la función o ya ha terminado y pertenece al código principal fuera de ese método).

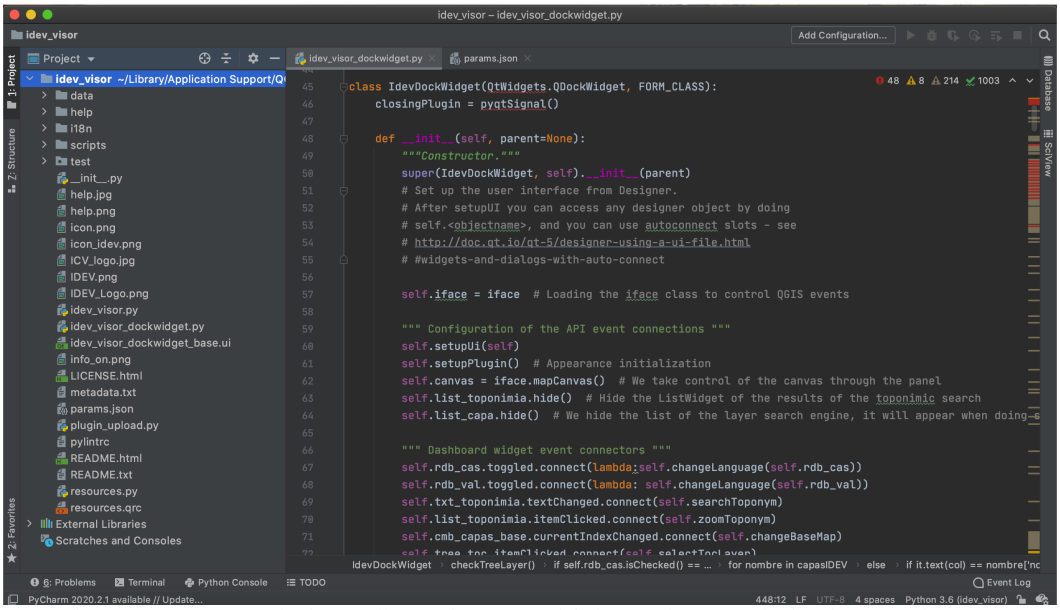


Ilustración 34. - IDE de desarrollo PyCharm con el fichero "ide_visor_dockwidget.py" abierto.

Todos los ficheros necesarios para crear el complemento ya se encuentran dentro de la carpeta indicada anteriormente y que el "*Plugin Builder*" ha creado. En este caso, todo el código fuente se ha escrito en el fichero "*ide_visor_dockwidget.py*" ya que se corresponde con el panel que contiene todos los *widgets.* Es en este fichero donde se realizan las conexiones a los eventos de estos componentes y que desencadenan las diferentes funcionalidades, como por ejemplo, el cambio de idioma al pulsar un *RadioButton* o cargar una capa base al elegir una del desplegable correspondiente.

El fichero "*idev_visor.py*" es el fichero principal del componente en cuanto a configuración. Todos los componentes tienen este fichero que se denomina como el complemento, en este caso se denomina "*idev_visor*". Este parámetro se indica al crear el complemento y como vemos dentro de la capeta que contiene todos los complementos instalados en QGIS, tiene ese mismo nombre, es decir, la carpeta que contiene el *plugin* se denomina "*idev_visor*". En el se crea la clase principal de la que dependen todos los demás objetos, en este caso se ha denominado "*Idev*".

En este fichero se define la clase principal "Idev", al definirla, se realiza una instancia a la clase "*iface*" que le proporciona todos los métodos para manipular QGIS y el proyecto de trabajo durante la ejecución (capas, canvas, configuración, eventos,…). La clase "iface" dentro de QGIS es del tipo "*QgsInterface*". Permite acceder a la mayoría de los componentes gráficos utilizados en *QGIS* permitiendo acceder a sus métodos y propiedades. Esta librería permite controlar por código casi todas las acciones que se pueden realizar dentro del programa aunque su uso más habitual es el control del canvas (carga de capas, control de la visualización, consulta a objetos, …).

En resumen, en este complemento hacemos uso de *Qt* (kit de widgets para crear una interfaz en *Python), PyQt* (es la interfaz de *Python* para *Qt)* y *PyQGIS* (API de *QGIS* para *Pyhton* y que se integra dentro de *PyQt).*

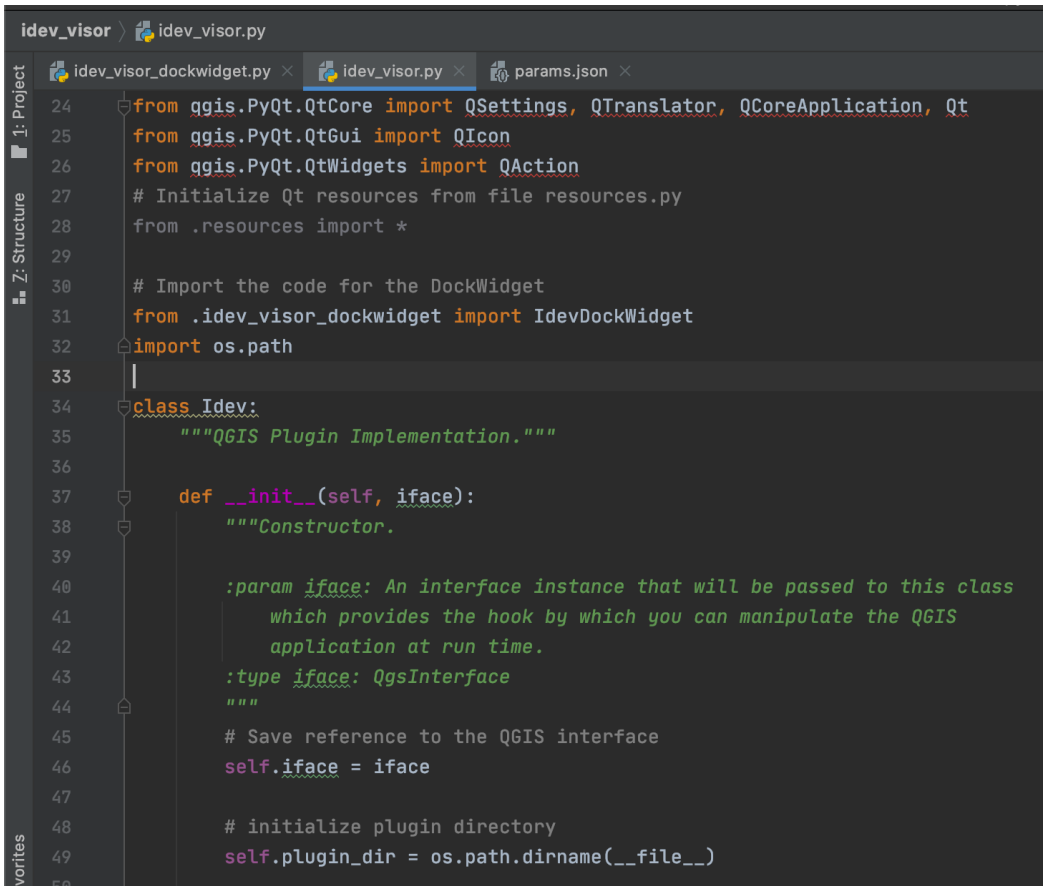


Ilustración 35. - Definición e inicialización de la clase "idev", clase principal del complemento.

Veamos ahora detalladamente la programación del complemento.

1.4. Configuración inicial del complemento.

Este complemento por lo tanto, se basa en dos tipos principales de componentes, los que proporciona la librería *PyQt* y los que proporciona *PyQGIS*. Al principio de cada fichero .py se hace una importación de todos los componentes que se van a utilizar, tanto los propios de Python como los de QGIS. Los genéricos de *Python* son *PyQt5* (versión utiliada) y la *API* de *QGIS* se invocan con la librería *qgis.* En este caso se han utilizado los siguientes componentes (se puede consultar al comienzo del fichero "*idev_visor_dockwidget.py*":

```
import os
import sys
import webbrowser
from PyQt5 import Qt, uic, QtGui, QtCore
from PyQt5.QtWidgets import QTreeWidgetItem, QMessageBox
from qgis.PyQt import *
from qgis.PyQt.QtCore import *
from qgis.utils import iface
from qgis.core import QgsRectangle, QgsRasterLayer, QgsProject,
QgsVectorLayer, QgsLayerTreeGroup, QgsLayerTreeLayer, \
    QgsLayerTreeNode, QgsGeometry, QgsPointXY, QgsPoint
from PyQt5.QtGui import QPixmap
import requests, json
from json import loads
```

```
from qgis.PyQt.QtCore import QUrl
from qgis.PyQt.QtWebKitWidgets import QWebView
```

Una vez se han dispuesto los diferentes componentes con el programa "*Qt Designer*" como se ha visto anteriormente, al comienzo de la definición del panel (que en este caso se define como una clase con el nombre "*IdevDockWidget*") y que carga la configuración del fichero "*idev_visor_dockwidget_base.iu*" ya descrito.

```
FORM_CLASS, _ = uic.loadUiType(os.path.join(
    os.path.dirname(__file__), 'idev_visor_dockwidget_base.ui'))
```

Seguidamente se implementa la clase "*IdevDockWidget"* desde la que se controla todas las funcionalidades y enlaza los widgets con los eventos de QGIS (como cargar, descargar capas o centrar el zoom en un encuadre determinado). Se comienza con la definición de la clase y a continuación se incluye la función "__init__", función clave en todas las clases de Python y que siempre se ejecuta cuando se instancia al comienzo de la ejecución.

```
class IdevDockWidget(QtWidgets.QDockWidget, FORM_CLASS):
    closingPlugin = pyqtSignal()

    def __init__(self, parent=None):
        """Constructor."""
        super(IdevDockWidget, self).__init__(parent)
        # Set up the user interface from Designer.
        # After setupUI you can access any designer object by doing
        # self.<objectname>, and you can use autoconnect slots - see
        # http://doc.qt.io/qt-5/designer-using-a-ui-file.html
        # #widgets-and-dialogs-with-auto-connect

        self.iface = iface  # Loading the iface class to control QGIS
events
```

Primeramente se invocan unas funciones de configuración que cambian el aspecto del complemento y cargan los elementos del mismo. También lo conecta a la interface "iface" para poder invocar las acciones de QGIS.

```
      """ Configuration of the API event connections """
self.setupUi(self)
self.setupPlugin()              #       Appearance       initialization
self.canvas = iface.mapCanvas()  # We take control of the canvas through
the                                                                panel
self.list_toponimia.hide()  # Hide the ListWidget of the results of the
toponimic                                                        search
self.list_capa.hide()  # We hide the list of the layer search engine,
it will appear when doing search

""" Layer QTreeWidget Settings  """

self.tree_toc.headerItem().setText(0, "")

""" We dynamically load the layers in the QtreeWidget at various
parent / child levels """

self.createTree()
```

La función "*setupUI*" y "*setupPlugin*" sirven para la configuración inicial del visor, se establece un idioma y los iconos e imágenes que se cargan. Para ello se asigna un widget tipo *QPixmap* o *QIcon* a cada uno de los elementos. También oculta las listas en las que se muestran los resultados de la búsqueda toponímica y de capas, cuando se utilizan se hacen visibles mostrando los resultados.

```
""" Function to load the initial appearance of the plugin  """
def setupPlugin(self):
    baseDirectory = os.path.dirname(os.path.realpath(__file__)) #
Plugin absolute path
    self.im_icv = QPixmap(os.path.dirname(os.path.realpath(__file__))
+ '/IDEV.png')
    self.txt_logo_2.setPixmap(self.im_icv)    # Loading the IDEV logo
    self.rdb_cas.setChecked(True)    # Default language "Castellano"

self.btn_meta.setIcon(QtGui.QIcon(os.path.dirname(os.path.realpath(__f
ile__)) + '/info_on.png'))

self.btn_icv.setIcon(QtGui.QIcon(os.path.dirname(os.path.realpath(__fi
le__)) + '/icon.png'))

self.btn_idev.setIcon(QtGui.QIcon(os.path.dirname(os.path.realpath(__f
ile__)) + '/icon_idev.png'))

self.btn_help.setIcon(QtGui.QIcon(os.path.dirname(os.path.realpath(__f
ile__)) + '/help.png'))
```

Una vez tenemos todos los *widgets* cargados en el panel y configurada su apariencia, se invoca la función "*createTree"* que en tiempo de ejecución, genera el árbol de capas siguiendo la misma estructura de padres/hijos que en el panel de *Capas* del visor de la *IDEV.* Debido a su extensión, únicamente referimos los tres casos que se producen al crear cada elemento del árbol.

```
def createTree(self):
    self.tree_toc.clear() # Clear the tree
    carto = QTreeWidgetItem(self.tree_toc)  # Create the QtreeWidget
    if self.rdb_cas.isChecked() == True: # Create the fahther and
child nodes in spanish
```

Con esto creamos el *QTreeWidegt* y lo inicializamos. Cada vez que se cambia el idioma se vuelve a invoca esta función ya que dependiendo del lenguaje seleccionado, carga unos textos de ítem u otros (los textos de ítem se corresponden con el nombre de las capas descritas en el fichero "*params.json*" y lo utilizamos para cargar/descargar datos o invocar la información del catálogo referente a esa capa.

- Nodo padre sin *checkbox:*

```
basecartografica05 = QTreeWidgetItem(carto)
basecartografica05.setText(0, "Base Cartográfica CV05")
```

Únicamente se le pone un nombre, haciendo referencia al ítem padre principal que en ese caso se ha denominado "*carto*" y del que colgarán el resto de nodos hijo.

- Nodo padre con *chechbox*

```
orografia05 = QTreeWidgetItem(basecartografica05)
orografia05.setFlags(orografia05.flags() | Qt.ItemIsTristate |
Qt.ItemIsUserCheckable)
orografia05.setText(0, "Orografía")
orografia05.setCheckState(0, Qt.Unchecked)
```

Este siguiente nodo es hijo de "carto" pero a su vez será padre de otros (en este caso, de todas las capas que configuran la orografía en la serie BCV05). En este caso si que se define como un "*checkbox"* y que aparece "*unchecked"*.

- Nodo hijo (siempre con *checkbox)*

```
puntoscota = QTreeWidgetItem(orografia05)
puntoscota.setFlags(puntoscota.flags() |
Qt.ItemIsUserCheckable)
puntoscota.setText(0, "Puntos de cota CV05")
puntoscota.setCheckState(0, Qt.Unchecked)
```

En este caso es exclusivamente nodo hijo de "*orografia05*" y no cuelga de el ninguno más (no tiene la propiedad "*Qt.ItemIsTristate*"). Es también del tipo "*checkbox"* y el texto de este ítem es "*Puntos de cota CV05*". Esta propiedad es la que utilizamos para conectar con el fichero "*params.json*", veamos como está definido este ítem en ese fichero:

```
{
  "nombre_cas": "Puntos de cota CV05",
  "nombre_val": "Punts de cota CV05",
  "aviso_cas": "",
  "aviso_val": "",
  "metadato": "spaicvBcv05Serie2015",
  "url":
"crs=EPSG:25830&dpiMode=7&format=image/png&layers=OrogpuntosCV05&style
s&url=http://terramapas.icv.gva.es/mapa_topografico_base"
}
```

Cada ítem consta de seis parámetros que utilizaremos para la carga en la tabla de contenidos y para enlazar con el catálogo de la IDEV a través del identificador de metadato.

- "nombre_cas": Nombre de la capa en castellano con en el que se carga en la tabla de contenidos y aparece en el *QTreeWidget*.
- "nombre_val": Nombre de la capa en valenciano con en el que se carga en la tabla de contenidos y aparece en el *QTreeWidget*
- "aviso_cas": Texto en castellano que aparece cuando se carga la capa, en el caso que no sea nulo.
- "aviso_val": Texto en valenciano que aparece cuando se carga la capa, en el caso que no sea nulo.
- "metadato": identificador único del metadato de la capa y que sirve para invocar la página web de la interfaz del catálogo y que da acceso a la información del metadato y a los servicios de descarga.
- "url": Cadena de conexión que sirve para cargar en QGIS la capa correspondiente a través de su servicio WMS de visualización.

Una vez cargada la interfaz y todos los controles en el panel, hay que conectar los eventos de los mismos con las funciones que realizan las acciones deseadas en QGIS. Esto se realiza a continuación y dentro de esta función. Para ello se utiliza la siguiente sintaxis:

```
self.txt_toponimia.textChanged.connect(self.searchToponym)
```

En la anterior sentencia la palabra "*self*" hace referencia a si mismo, es decir, al panel y por consiguiente "*self.txt_toponimia*" se refiere un componente tipo QLineEdit que es la caja de texto donde se escriben los caracteres para la búsqueda toponímica. A continuación se hace referencia al evento de ese componente que desata la función, en este caso es "*textChanged*". Este evento se desencadena según va cambiando el texto de la búsqueda, es decir, cada vez que se añade o quita un carácter invoca la función "*searchToponym*". Después del evento que queremos detectar viene la sintaxis "*.connect*" que tiene entre paréntesis la función que invoca, como se ha indicado anteriormente es "*self.searchToponym*". Se invoca con el self. Porque está definida en el mismo panel contenedor de los componentes. Más adelante describiremos todas las funciones programadas.

Los eventos que se han capturado cuando se utiliza el complemento son los indicados a continuación, siguen la sintaxis anteriormente indicada y se incluyen en la definición principal del panel.

```
""" Dashboard widget event connectors """
self.rdb_cas.toggled.connect(lambda:self.changeLanguage(self.rdb_cas))
self.rdb_val.toggled.connect(lambda:
self.changeLanguage(self.rdb_val))
self.txt_toponimia.textChanged.connect(self.searchToponym)
self.list_toponimia.itemClicked.connect(self.zoomToponym)
self.cmb_capas_base.currentIndexChanged.connect(self.changeBaseMap)
self.tree_toc.itemClicked.connect(self.selectTocLayer)
self.tree_toc.itemChanged.connect(self.checkTreeLayer)
self.txt_busca_capa.textChanged.connect(self.searchLayer)
self.list_capa.itemClicked.connect(self.selectTreeLayer)
self.btn_meta.clicked.connect(self.openCatalog)
self.btn_icv.clicked.connect(self.openIcvSite)
self.btn_idev.clicked.connect(self.openIdevSite)
self.btn_help.clicked.connect(self.openHelp)
```

Veamos a continuación las principales funcionalidades del complemento y como son las funciones que las implementan, en la siguiente figuran se ha enmarcado el *widget* y la función que se desencadena al usarlo y que más adelante se comenta el código generado:

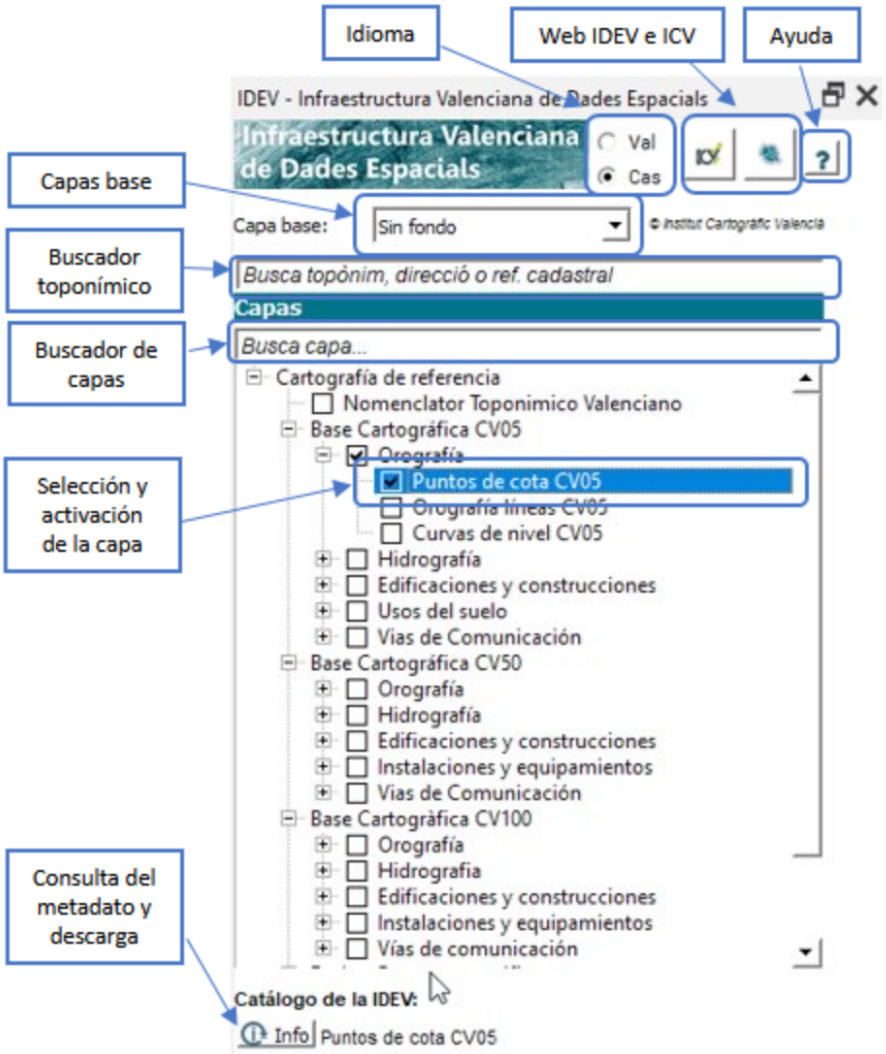


Ilustración 36. - Funcionalidades principales del complemento en la interfaz.

1.5. Cambio de idioma.

Una vez cargado el panel y todos los componentes, por defecto se cargan todos los textos en castellano y el control "*rdb_cast*" que se corresponde con el *RadioButton* que traduce al castellano el panel, aparece activado por defecto. El otro control "*rdb_val*" sirve para traducir todos los elementos al valenciano. La función que se invoca al activar cada uno de estos controles es "*changeLanguage*" y realiza varias acciones. Crea de nuevo el árbol de capas en el idioma seleccionado, traduce los textos del panel y los "*tooltips*" de ayuda de los controles. Se adjunta la primera parte de la función en que hace estos cambios generales y los particulares para el idioma seleccionado. Esta función, en la invocación, se le pasa como parámetro el propio control, la variable utilizada es "*b*" y se utiliza el texto del control ("*cas*" o "*val*") para desencadenar la traducción pertinente.

Para el acrónimo del lenguaje, los correcto hubiera sido elegir el código ISO correspondiente para cada idioma, en el caso del castellano se utiliza "*-es*" y para el valenciano se usa "*-cat*", pero hemos utilizado los mismos que constan en el visor de la IDEV que se refiere a los mismos como "-cas" y "-val". Si usáramos el mecanismo de identificación del lenguaje del sistema operativo tendríamos que usar la nomenclatura especificada en la ISO. También se ha utilizado la identificación "-val" porque los

servicios utilizados como en *Nomenclátor* de la *Academia Valenciana de la Lengua*, utilizan esta lengua y no el catalán.

```
def changeLanguage(self, b):
    self.createTree()  # Function that changes the language of the
controls. The languages are Spanish / Valencian
    self.txt_capa_seleccionada.setEnabled(False)  # Activate the label
of the active layer
    self.txt_capa_seleccionada.setText("")  # Claer the text of the
label layer name
    self.txt_info_capa.setEnabled(False)  # Desactivate the label of
the layer name
    self.btn_meta.setEnabled(False)  # Desactivate the button of the
info layer
    """ Check the language and set the text of the widgets """
    if b.text() == "Cas":
        if self.rdb_cas.isChecked():
            self.txt_toponimia.setText("Busca topónimo, dirección o
ref. catastral")
            self.txt_busca_capa.setText("Busca capa...")
            self.cmb_capas_base.setItemText(0, "Sin fondo")
            self.cmb_capas_base.setItemText(1, "Imagen")
            self.cmb_capas_base.setItemText(2, "Híbrido")
            self.cmb_capas_base.setItemText(3, "Relieve")
            self.cmb_capas_base.setItemText(4, "Topográfico")
            self.cmb_capas_base.setItemText(5, "Topográfico gris")
            self.cmb_capas_base.setItemText(6, "Topográfico básico")
            self.txt_info_capa.setText("Catálogo de la IDEV:")
            """ Tooltilps translations to Spanish """
            self.btn_icv.setToolTip("Ir a la web del Institut
Cartogràfic Valencià")
            self.btn_idev.setToolTip("Ir a la web de la
Infraestructura Valenciana de Datos Espaciales")
            self.cmb_capas_base.setToolTip("Seleccionar capa base de
fondo")
            self.txt_toponimia.setToolTip("Inserte el texto para la
búsqueda toponímica")
            self.btn_meta.setToolTip("Pulsar para consultar el
catálogo de la IDEV de la capa seleccionada")
            self.rdb_cas.setToolTip("Activar para traducir el
complemento al castellano")
            self.rdb_val.setToolTip("Activar para traducir el
complemento al valenciano")
            self.txt_busca_capa.setToolTip("Inserte el texto para la
búsqueda de capas")
```

Se puede cambiar en tiempo de ejecución el idioma sin tener que recargar el complemento. Otra posibilidad para hacer el complemento multilingüe es utilizar las propiedades que ofrece Qt "*QtLinguist*" y que sirve para asignar a cada texto del componente una traducción en un idioma. Esto es útil para instalar el componente o que se cargue, en la misma lengua local en la que se tenga configurado el sistema operativo, o para poder elegir en el instalador de complementos de QGIS que lenguaje usar, pero no permite hacer el cambio en tiempo de ejecución.

Cada uno de los dos sistemas (traducir en tiempo de ejecución o utilizando el *QtLinguist)* tiene sus ventajas e inconvenientes. En este trabajo se ha optado por la traducción en tiempo de ejecución.

1.6. Invocación de las webs de la IDEV y del ICV.

Desde dentro del panel a través de dos *QPushButton* se puede abrir la web de la "Infraestructura Valenciana de Datos Espaciales" y la del "Institut Cartogràfic Valencià". Ambos botones usan el evento "*clicked*" para invocar las funciones de apertura de estos sitios web. Para ello se utiliza la funcionalidad de Python "*webbrowser*". Este sería el código fuente de una función (es igual en ambos casos):

```
""" Button to open the ICV website   """

def openIcvSite(self):
    if self.rdb_cas.isChecked() == True:
        webbrowser.open('http://www.icv.gva.es/es/inicio')
    if self.rdb_val.isChecked() == True:
        webbrowser.open('http://www.icv.gva.es/va/inicio')
```

1.7. Invocación de la ayuda del complemento.

Se ha incluido una ayuda para el manejo del complemento que se invoca por medio de un *QPushButton* de la misma manera que en el caso anterior. En esta ocasión, la página web que se invoca está en local, dentro del directorio "*data*" en los archivos del complemento. Hay un manual para cada idioma, la función "*openHelp*" detecta el idioma en el que está en ese momento el complemento configurado y lanza la web correspondiente. En los anexos de este trabajo se incorpora la ayuda en castellano.

```
def openHelp(self):
    if self.rdb_cas.isChecked() == True:
        webbrowser.open(os.path.dirname(os.path.realpath(__file__)) +
'/data/help_cast.html')
    if self.rdb_val.isChecked() == True:
        webbrowser.open(os.path.dirname(os.path.realpath(__file__)) +
'/data/help_val.html')
```

1.8. Selector de capa base.

Al igual que en el visor de la IDEV, se puede seleccionar la capa base sobre la que trabajar en el proyecto de QGIS. Son seis capas las que se pueden elegir, la séptima opción es "Sin capa base" que eliminaría del proyecto las capas base cargadas.

Para activar una capa base se tiene que elegir directamente del desplegable la opción deseada. Pulsando sobre un ítem del *ComboBox* se desencadena la función "*changeBaseMap"* que añade a la tabla de contenidos las capas correspondientes. En la mayoría de capas base se usan servicios WMTS cacheados que se visualizan muy rápidamente y se acompañan con el límite de la comunidad autónoma. Lo primero que hace la función es almacenar en una lista el nombre de las capas que hay en el proyecto, luego recorre esta lista y descarga del proyecto las capas que se encuentran entre las capas base. Posteriormente carga las capas base elegidas del desplegable. Este comportamiento se justifica porque puede haber capas base guardadas en el proyecto de QGIS y es necesario quitarlas cuando se cambia de capa base, también cuando se elige la opción de "*Sin capas base*" y no se elige ninguna, en este caso se descargan del proyecto y no se carga ninguna.

```
def changeBaseMap(self):
    extent = self.canvas.extent()  # Store the actual extent in a
variable
    names = [layer.name() for layer in
QgsProject.instance().mapLayers().values()]  # Store in a list the
names of the layers
    for i in names:  # Unload the layer by name
        if i == 'Contorno autonómico':
            self.unloadLayerToc('Contorno autonómico')
        if i == 'Ortofoto 2019':
            self.unloadLayerToc('Ortofoto 2019')
        if i == 'ESRI Satellite':
            self.unloadLayerToc('ESRI Satellite')
        if i == 'Capa base Híbrido':
            self.unloadLayerToc('Capa base Híbrido')
        if i == 'Capa base Relieve':
            self.unloadLayerToc('Capa base Relieve')
        if i == 'Capa base Topográfico':
            self.unloadLayerToc('Capa base Topográfico')
        if i == 'Capa base Topográfico Básico':
            self.unloadLayerToc('Capa base Topográfico Básico')
        if i == 'Capa base Topográfico Gris':
            self.unloadLayerToc('Capa base Topográfico Gris')
```

La descarga de las capas se realiza mediante la función *"unloadLayerToc"* a la que se pasa por parámetro el nombre de la capa de la tabla de contenidos que se quiere descargar. Esta función se invoca tanto desde esta parte del programa como desde el árbol de contenidos al desactivar un ítem del mismo. De igual manera hay una función de carga que se explica más adelante. La función de descarga es:

```
def unloadLayerToc(self, nombre):

QgsProject.instance().removeMapLayer(QgsProject.instance().mapLayersBy
Name(nombre)[0])  # Unload the layer form the TOC
    iface.mapCanvas().refresh()  # Refresh the projet view in QGIS
```

Para cargar una capa base, vemos por ejemplo, las capas que conforman el grupo "*Híbrido"*, en este caso se cargan las capas de imagen (ortofoto del año 2019, ortofoto de ESRI para visualizarse fuera de los límites de la Comunitat Valenciana, el Nomenclátor y la capa de línea limite). Este sería el código correspondiente que según el lenguaje seleccionado le otorga un nombre de capa u otro, la fuente de conexión es la misma en todos los casos. Se desencadena este bucle si en el desplegable el ítem seleccionado es el tercero, como se comienza a contar incluyendo el "0" este tercer ítem se corresponde con el numeral "2".

```
if self.cmb_capas_base.currentIndex() == 2:
    urlWithParams_hibrido =
'crs=EPSG:3857&dpiMode=7&format=image/png&layers=mapabase_hibrid&style
s&tileMatrixSet=mapabase_hibrid-wmsc-
0&url=http://terramapas.icv.gva.es/mapabase_hibrid/'
    rasterLyr_hibrido = QgsRasterLayer(urlWithParams_hibrido, 'Capa
base Híbrido', 'wms')
    if not rasterLyr_hibrido.isValid():
        print("Capa no válida")
        iface.messageBar().pushMessage("Error", "Capa no válida",
level=Qgis.Critical, duration=5)
    else:
        QgsProject.instance().addMapLayer(rasterLyr_hibrido, False)
```

```
        layerTree = iface.layerTreeCanvasBridge().rootGroup()
        layerTree.insertChildNode(-1,
QgsLayerTreeLayer(rasterLyr_hibrido))
    urlWithParams_orto =
'crs=EPSG:3857&dpiMode=7&format=image/png;%20mode%3D8bit&layers=01_8bi
ts_01_RGB_05_PNG&styles&tileMatrixSet=01_8bits_01_RGB_05_PNG-wmsc-
0&url=http://terramapas.icv.gva.es/odcv05_etrs89h30_2019_3857'
    rasterLyr_orto = QgsRasterLayer(urlWithParams_orto, 'Ortofoto
2019', 'wms')
    if not rasterLyr_orto.isValid():
        print("Capa no válida")
        iface.messageBar().pushMessage("Error", "Capa no válida",
level=Qgis.Critical, duration=5)
    else:
        QgsProject.instance().addMapLayer(rasterLyr_orto, False)
        layerTree = iface.layerTreeCanvasBridge().rootGroup()
        layerTree.insertChildNode(-1,
QgsLayerTreeLayer(rasterLyr_orto))
    urlWithParams_esri =
'type=xyz&zmin=0&zmax=20&url=https://server.arcgisonline.com/ArcGIS/re
st/services/World_Imagery/MapServer/tile/{z}/{y}/{x}'
    rasterLyr_esri = QgsRasterLayer(urlWithParams_esri, 'ESRI
Satellite', 'wms')
    if not rasterLyr_esri.isValid():
        print("Capa no válida")
        iface.messageBar().pushMessage("Error", "Capa no válida",
level=Qgis.Critical, duration=5)
    else:
        QgsProject.instance().addMapLayer(rasterLyr_esri, False)
        layerTree = iface.layerTreeCanvasBridge().rootGroup()
        layerTree.insertChildNode(-1,
QgsLayerTreeLayer(rasterLyr_esri))
```

Las capas siempre se cargan al final de la tabla de contenidos ya que no deben de superponerse con las del resto del proyecto, por lo que en la instrucción de carga se le pasa como parámetro el valor "-1" que la añade la última, en el ejemplo se ve como se añade el servicio WMTS de ESRI cuya cadena de conexión está almacenada en la variable *"rasterLyr_esri":*

```
layerTree.insertChildNode(-1, QgsLayerTreeLayer(rasterLyr_esri))
```

1.9. Buscador toponímico.

La funcionalidad del buscador toponímico se hace a través de una entrada texto que se encuentra en la interfaz y que tiene el texto de ayuda "*Busca topónimo, dirección o refer. catastral…*", al teclear sobre este control del tipo *QLineEdit* se ha conectado el evento *"textChanged"* que invoca la función "*searchToponym*". Esta función recoge el texto que se va tecleando y cuando es mayor de tres caracteres realiza la petición al servicio de "*Solr*" de la *IDEV*, que devuelve los topónimos encontrados, el tipo de topónimo, su clasificación y el *extent* del mismo a través de sus coordenadas máximas y mínimas.

En primer lugar se comprueba que el texto del control no es el texto de ayuda, si no lo es, continua con la ejecución de la función.

```
""" Function for toponymy search. Connection with the IDEV search
engine service """

def searchToponym(self):
    # Disable the search with the default help text of the control
    if self.txt_toponimia.text() == 'Busca topónimo, dirección o ref.
catastral' or self.txt_toponimia.text() == 'Busca topònim, direcció o
ref. cadastral':
        pass
    # When change the text of the textline activate the search
```

Una vez se ha introducido un patrón de texto, cuando la longitud es mayor de tres, se realiza una petición http al servicio de búsqueda toponímica de la IDEV, se utiliza el componente de *Python* "*request*" que almacena en una variable la respuesta de la petición. En este caso lo almacenamos en la variable "s", se construye primeramente la cadena de consulta (almacenada en la variable "*url*") y luego se invoca el método "*get*" del componente "*request*".

```
    else:
        text = self.txt_toponimia.text()
        self.list_toponimia.clear()
        if len(text) > 2:  # The search began when type 3 or more
characters
            self.list_toponimia.show()  # Show the textlist of the
results of the search
            url =
'http://descargas.icv.gva.es/server_api/buscador/solrclient.php?start=
0&limit=40&query=' + text   # Create the url search query
            s = requests.get(str(url)).text  # Request the query
```

Un ejemplo de petición, por ejemplo, para el patrón de texto "*Cullera*", sería el siguiente:

http://descargas.icv.gva.es/server_api/buscador/solrclient.php?start=0&limit=40&query=Cullera

La respuesta del servicio es un fichero JSON con la siguiente información:

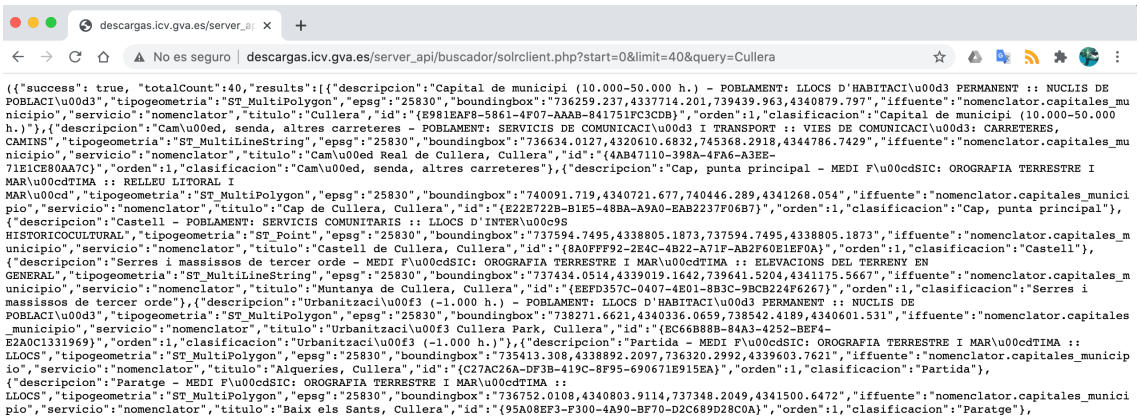


Ilustración 37. - Respuesta de una búsqueda toponímica en el servicio Solr de la IDEV.

Es un listado extenso con todas las respuestas ordenadas por mayor coincidencia y limitadas a 40 elementos. El siguiente paso que realiza el algoritmo es extraer de toda la respuesta, únicamente el texto correspondiente a la estructura *JSON* que contiene los

topónimos encontrados junto con sus propiedades, para ello recorre todo el texto hasta encontrar el primer carácter *"["* que marca el comienzo de esa información hasta el final marcado con un carácter *"]"*. Toda esa cadena de texto se guarda en la variable "*substring*".

```
            start = s.find("[") + len("[")
            end = s.find("]")
            substring = "[" + s[start:end] + "]" # Format of the
response. Create a JSON file
            # print(substring)
```

Una vez tenemos la secuencia complete aislada, cargamos los datos en un diccionario en la variable *"y"* ya que al utilizar la función *"json.loads"* lo crea automáticamente. De todos los campos que contiene cada ítem, únicamente añadimos al *QlistWidget* los siguientes:

- Titulo.- Nombre del topónimo.
- Clasificación. – Clasificación del topónimo según el Nomenclátor oficial de la *Academia Valenciana de la Lengua.*
- Descripción. – Descripción mas detallada del tipo de topónimo.

```
            y = json.loads(substring)  # Load the result as a JSON
file
            i = len(y)  # Calculate the lenght of the query result
            for x in range(0, i):   # Load in the ListWidget the
results of the query
                titulo = y[x]["titulo"]
                clasificacion = y[x]["clasificacion"]
                descripcion = y[x]["descripcion"]
                if titulo.find('\n') != -1:
                    titulo = titulo.replace('\n', '')  # remove that
character of some query results
                if descripcion == '':
                    descripcion = ''
                else:
                    descripcion = "\n" + descripcion  # Format the
result in the ListWidget
```

En la siguiente figura vemos la respuesta que genera el buscador de *Solr* formateada.

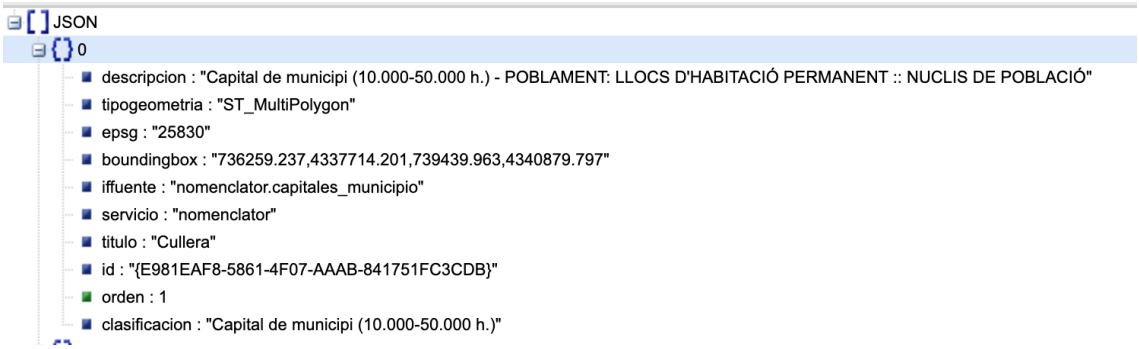


Ilustración 38.- JSON de respuesta a la búsqueda de "Cullera" del servicio toponímico de Solr.

Estos ítems se muestran en el desplegable "*list_toponimia*" con el siguiente código. Una vez se muestran se pueden hacer clic sobre ellos y te centra la vista en el encuadre del topónimo seleccionado.

```
                widgetText =
QtWidgets.QListWidgetItem('{0}'.format(titulo) + "\t " + clasificacion
+ descripcion)
                self.list_toponimia.addItem(widgetText)  # Add items
to the ListWidget
        else:
            self.list_toponimia.hide()  # If the query is null, hide
the ListWidget
```

Esta sería lo que se nos muestra en el complemento a la búsqueda del ítem "*Cullera*", aparece en primer lugar la mejor coincidencia:

Ilustración 39. - Resultado del buscador toponímico al buscar "Cullera".

Para ello se hace uso de la función "*zoomToponym"* de la que de adjunta el código fuente. Esta función aísla el ítem de la petición sacando la propiedad "*bounding box*" que contiene las coordenadas del topónimo y que almacenamos en cuatro variables, "*XMin, Xmax, Ymin, Ymax*". En el caso de que sea un topónimo puntual (que su coordenada *Xmin = Xmax*), en ese caso se muestra a una escala fija 1:1.000

```
""" Function that centers the zoom on the toponym selected from the
ListWidget  """

def zoomToponym(self):
    texto = self.list_toponimia.currentItem().text()  # Capture the
exact name of the toponym
    text, tipo = texto.split("\t")  # Split the information of the
toponym
    self.txt_toponimia.setText(text)  # Write the toponymin the
labeltext of search
    text = text.replace('/', '')  # Format the toponym text
    url =
'http://descargas.icv.gva.es/server_api/buscador/solrclient.php?start=
0&limit=40&query=' + text  # Create the query with the exact toponym
    s = requests.get(str(url)).text
```

```
        start = s.find("[") + len("[")
        end = s.find("]")
        substring = "[" + s[start:end] + "]"  # Format of the response.
Create a JSON file
        y = json.loads(substring)  # Load the result as a JSON file
        extent = y[0]["boundingbox"]  # Get the boundingbox of the toponym
        XMin, Ymin, XMax, Ymax = extent.split(",")  # Asign the
coordinates of the extent
        zoomRectangle = QgsRectangle(float(XMin), float(Ymin),
float(XMax), float(Ymax))  # Create a zoomRectangle
        self.iface.mapCanvas().setExtent(zoomRectangle)  # Set the extent
o the project
        if XMin == XMax:
            self.iface.mapCanvas().zoomScale(1000)  # Default scale if the
toponym is a puntual shape
        self.iface.mapCanvas().refresh()  # Refresh the canvas view
        self.list_toponimia.hide()  # Hide the listwidget
```

El buscador toponímico de la IDEV no sólo busca topónimos sino que tiene indexados las direcciones postales del proyecto RT (Red de transportes), las referencias catastrales de las parcelas y puntos de interés de la Comunitat Valenciana, como por ejemplo centros educativos, hospitales, centros de mayores, etc…

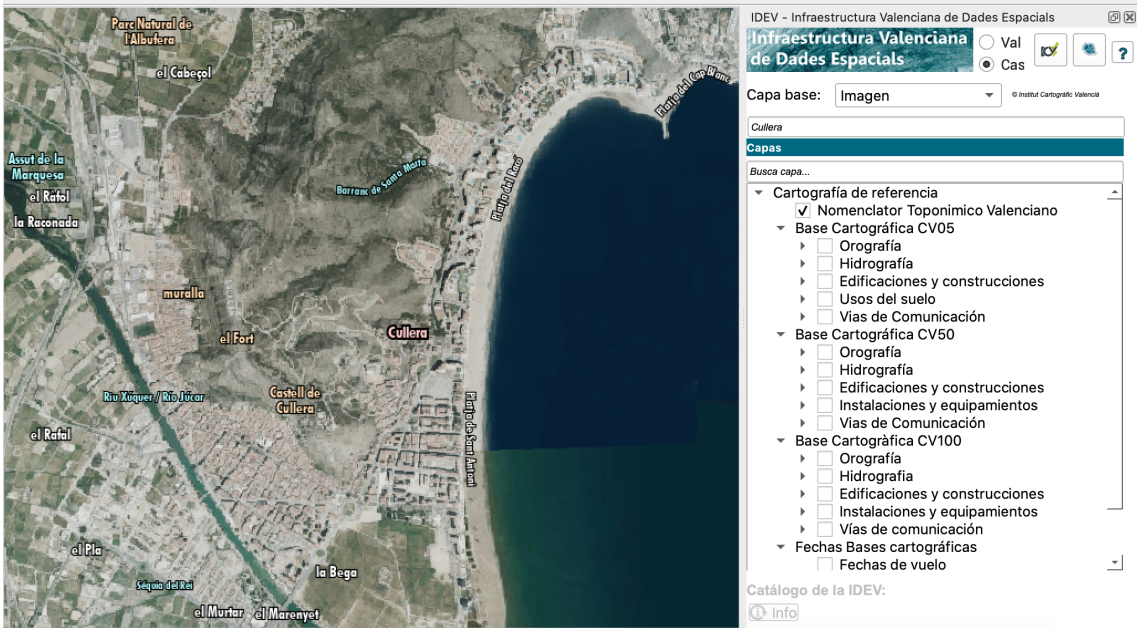


Ilustración 40. - Encuadre de QGIS al topónimo de "Cullera".

1.10. Buscador de capas.

Otra de las funcionalidades que presenta el visor de la IDEV y que facilita la navegación entre los datos y el descubrimiento de servicios cartográficos es el buscador de capas. A través de un *QListItem* (en este caso se denomina "*list_capa*") se muestra el resultado de la búsqueda de un patrón de texto que se introduce en un *QLineEdit* (en este caso se denomina "*txt_busca_capa*"). La función de búsqueda se denomina "*searchLayer*" y realiza la búsqueda sobre el diccionario de datos que contiene las capas. Para ello se carga en la variable "capasIDEV" toda la información del fichero "*params.json*" descrito anteriormente.

```
""" We define the dictionary of layers outside the functions so that
it is accessible in all """

path_capas_json = os.path.dirname(os.path.realpath(__file__)) +
'/params.json' # Path of the JSON file with the definition of each
layer
with open(path_capas_json, "r") as read_file:  # Read the JSON file
    capasIDEV = json.load(read_file)  # Store in a dictionary the JSON
file, each element is a layer
```

Se define fuera de la clase principal para que sea accesible desde todas las funciones y no únicamente desde un procedimiento. La función "searchLayer" recorre todos los elementos y devuelve aquellos en los que hay una coincidencia de texto en el nombre de la capa:

```
""" Layer search function in the QtreeWidget """

def searchLayer(self):
    if self.txt_busca_capa.text() == 'Busca capa...': # Help text
        pass
    else:  # Start the search if the text is not the help text
        text_capa = self.txt_busca_capa.text()  # Assign the search
text to the variable
        self.list_capa.clear()
        if len(text_capa) > 2:  # Star the search if the text lenght
is bigger than 3 characters
            self.list_capa.show()
            if self.rdb_cas.isChecked() == True: # Search in spanish
                i = 0
                for item in capasIDEV:  # We go through all the layers
                        if text_capa.lower() in
capasIDEV[i]['nombre_cas'].lower():

self.list_capa.addItem(capasIDEV[i]['nombre_cas']) # If a layer
matches the search pattern it is added to the listWisget
                        i = i + 1
            if self.rdb_val.isChecked() == True:  # Search in
valencian
                i = 0
                for item in capasIDEV: # We go through all the layers
                    if text_capa.lower() in
capasIDEV[i]['nombre_val'].lower():

self.list_capa.addItem(capasIDEV[i]['nombre_val']) # If a layer
matches the search pattern it is added to the listWisget
                    i = i + 1
        else:
            self.list_capa.hide() # Hide the listWidget of the results
of the layers search
```

Una vez se localizan las coincidencias, se muestran en un desplegable las capas candidatas, al pulsar sobre una de ellas se invoca la función "*selectTreeLayer*" que remarca sobre el árbol de capas la seleccionada. Este sería el código fuente:

```
""" Function that when selecting a layer of the QTreeWidget activates
the link button with the catalog """

#@QtCore.pyqtSlot(QtWidgets.QTreeWidgetItem, int)
def selectTreeLayer(self):
```

```
    self.tree_toc.clearSelection() # Clear the actual layers selection
    nom_capa = self.list_capa.currentItem().text() # We assign to the
variable the selected layer of the QTreeWidget
    for item in self.tree_toc.findItems(nom_capa, Qt.MatchContains |
Qt.MatchRecursive):
        self.tree_toc.setCurrentItem(item) # Capture of the selected
item by clicking
        item.setSelected(True) # Select the item in the QTreeWidget
        item.setExpanded(True) # Expand the QtreeWidget at the item
level
    if self.rdb_cas.isChecked() == True:
        self.txt_busca_capa.setText("Busca capa...")
    if self.rdb_val.isChecked() == True:
        self.txt_busca_capa.setText("Busca capa...")
    self.list_capa.hide()
```

Por ejemplo, si en la búsqueda introducimos el patrón de búsqueda "*CV05*" se muestran todas aquellas que contienen esos caracteres en su nombre. La búsqueda no distingue entre mayúsculas y minúsculas por lo que las coincidencias son mayores.

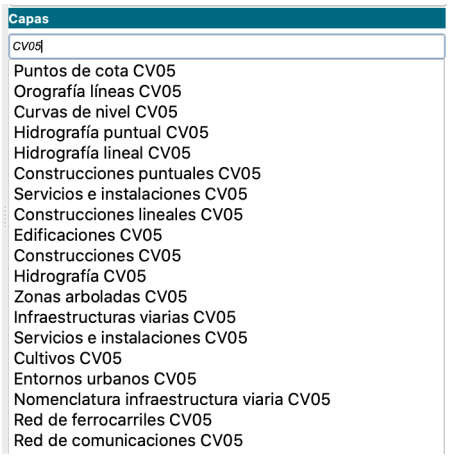


Ilustración 41. - Búsqueda de capas por el patrón "CV05".

Una vez se selecciona una de ellas, por ejemplo, "Puntos de cota CV05" vemos como se muestra en el árbol de capas con una selección en color gris claro. Si quisiéramos utilizarla ya podríamos activarla.

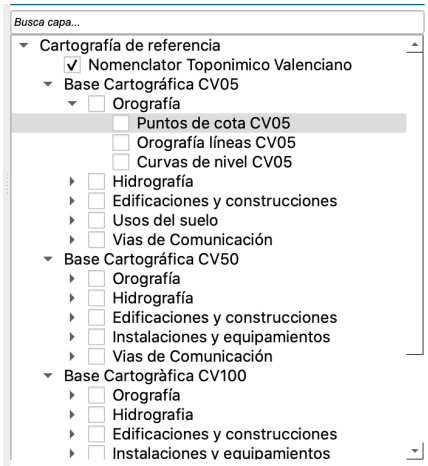


Ilustración 42.- Selección de la capa "Puntos de cota CV05".

1.11. Árbol de capas. Carga y descarga.

Cuando se carga el complemento, en tiempo de ejecución carga en un widget tipo *QTreeWidget* toda la estructura de capas. Ya se ha explicado anteriormente que la función que lo realiza es "*createTree*" y que en función del idioma, carga unos nombres de capa u otros. Una vez se ha cargado esta estructura, se utiliza la función "*checkTreeLayer*" para la carga de un servicio cartográfico a partir de la activación del *checkbox* correspondiente.

```
""" Load the layer in the QGIS project whrn check the item in the
QTreeWidget """

def checkTreeLayer(self, it, col): # The layer name is different in
each language, there is a different search in the layer dictionary
    if self.rdb_cas.isChecked() == True:
        for nombre in capasIDEV:  # Search in the layer dictionary by
spanish name
            if it.checkState(0) == QtCore.Qt.Checked:
                if it.text(col) == nombre['nombre_cas']:
                    self.loadLayerToc(nombre['url'],
nombre['nombre_cas'])  # load the layer Checked in the QGIS project
                    if nombre['aviso_cas'] != "":
                        msg = QMessageBox.information(self, "Aviso:",
nombre['aviso_cas'])  # Show advice if is not null
                        #msg.exec()
            else:
                if it.text(col) == nombre['nombre_cas']:
                    self.unloadLayerToc(nombre['nombre_cas']) # Unload
the layer of the proyect if is unchecked
```

Esta función recoje el ítem activado del árbol de capas e invoca a la función "*loadLayerToc*" pasándole por parámetro el nombre de la capa y la url de conexión. La función carga en la tabla de contenidos del proyecto de QGIS el servicio cartográfico y con el nombre de la capa que aparece en la estructura.

```
""" Funcion for load a layer in the QGIS project by name and url
connection string """

def loadLayerToc(self, url, nombre):
    layer = QgsRasterLayer(url, nombre, 'wms')  # Create a PyQt layer
    if not layer.isValid():
        print("Capa no válida")
        iface.messageBar().pushMessage("Error", "Capa no válida",
level=Qgis.Critical, duration=5)
    else:
        QgsProject.instance().addMapLayer(layer, False)  # Add the
layer in the project
        layerTree = iface.layerTreeCanvasBridge().rootGroup()  # Add
the layer on the top of the TOC
        layerTree.insertChildNode(0, QgsLayerTreeLayer(layer))
```

De igual manera a que cuando se activa el *checkbox* correspondiente de una capa, carga un servicio, al desactivarlo descarga la capa correspondiente de la tabla de contenidos. Para ello hace uso de la función "*unloadLayerToc*" , en este caso únicamente hay que pasarle por parámetro el nombre de la capa.

```
""" Funcion for unload a layer in the QGIS project by layers name """

def unloadLayerToc(self, nombre):

QgsProject.instance().removeMapLayer(QgsProject.instance().mapLayersBy
Name(nombre)[0])  # Unload the layer form the TOC
    iface.mapCanvas().refresh()  # Refresh the projet view in QGIS
```

1.12. Consulta del metadato y enlace a los servicios de descarga.

Otra de las funcionalidades del complemento y que sirve para integrar toda la información que hay en el visor de la IDEV es la posibilidad de poder consultar el catálogo para una capa determinada. Esta funcionalidad se invoca al seleccionar una capa en el árbol. No es necesario cargarla en la tabla de contenidos para que se pueda consultar dicha información.

Si queremos consultar la información referente, por ejemplo, a la capa "Nomenclátor Toponímico Valenciano" debemos de pulsar encima de dicha capa, veremos como se selecciona dibujándose un recuadro de color azul oscuro por debajo, es en ese momento cuando ese ítem del árbol tiene el foco. Una vez ocurre este evento, se invoca la función "*selectTocLayer*".

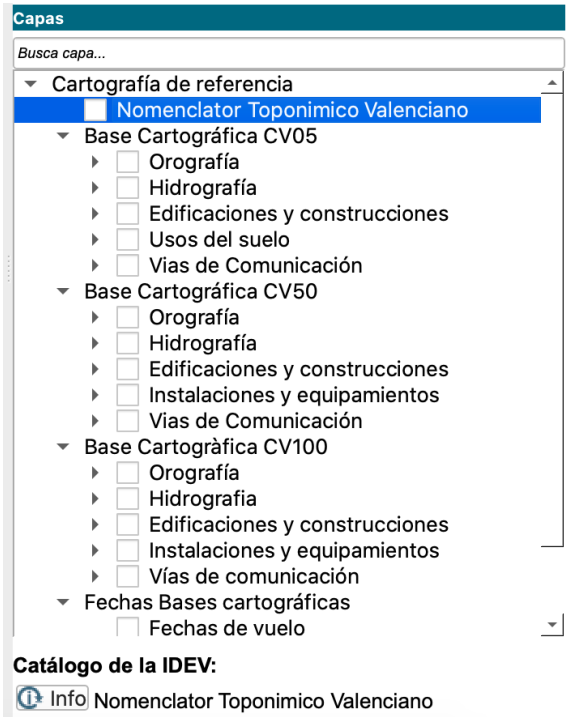


Ilustración 43. - Selección de una capa en el árbol de capas.

Esta función se desencadena al evento "*itemClicked*" para el widget "*QTreeWidget*" que alberga todos los ítems, una vez se ha clicado encima y se ha seleccionado, se ejecuta esta función.

```
def selectTocLayer(self):
    i = 0
    for ix in self.tree_toc.selectedIndexes():
        if self.rdb_cas.isChecked() == True:
            for nombre in capasIDEV:
```

```
                if ix.data() == nombre['nombre_cas'] and i == 0:
                    self.btn_meta.setEnabled(True)
                    self.txt_info_capa.setEnabled(True)

self.txt_capa_seleccionada.setText(nombre['nombre_cas'])
                    self.txt_capa_seleccionada.setEnabled(True)
                    i = i  + 1
```

Una vez se ha seleccionado la capa, se activa en la parte inferior del complemento el botón de consulta al metadato junto con el nombre de la capa que se puede consultar.

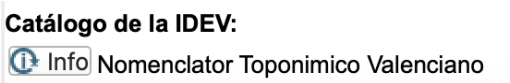


Ilustración 44. - Activación de la consulta al catálogo.

Pulsando sobre el botón de "*Info"* se abre en el navegador la interfaz del catálogo de la IDEV con la información del metadato, para ello se invoca la función "openCatalog" que usa el nombre de la capa para consultar en el diccionario de capas el identificador único del metadato. Recordemos que esta información se carga en la variable "*capasIDEV*" y que se lee del fichero "*params.json"*.

```
""" Button to open the catalog website of the selected layer in the
QTreeWidget """

def openCatalog(self):  # The layer name is different in each
language, there is a different search in the layer dictionary
    if self.rdb_cas.isChecked() == True:
        for nombre in capasIDEV:
            if self.txt_capa_seleccionada.text() ==
nombre['nombre_cas']:

webbrowser.open('http://www.icv.gva.es/auto/aplicaciones/icv_geocat/#/
search?uuid=' + nombre['metadato']) # Open the catalog
    if self.rdb_val.isChecked() == True:
        for nombre in capasIDEV:
            if self.txt_capa_seleccionada.text() ==
nombre['nombre_val']:

webbrowser.open('http://www.icv.gva.es/auto/aplicaciones/icv_geocat/#/
search?uuid=' + nombre['metadato']) # Open the catalog
```

Una vez clicado el botón se muestra en el navegador el catálogo de la IDEV. En esta información podemos acceder al metadato, a las *url* de conexión de los servicios cartográficos en que se sirve dicha capa y a los servicios de descarga en los formatos en que se ofrezca.

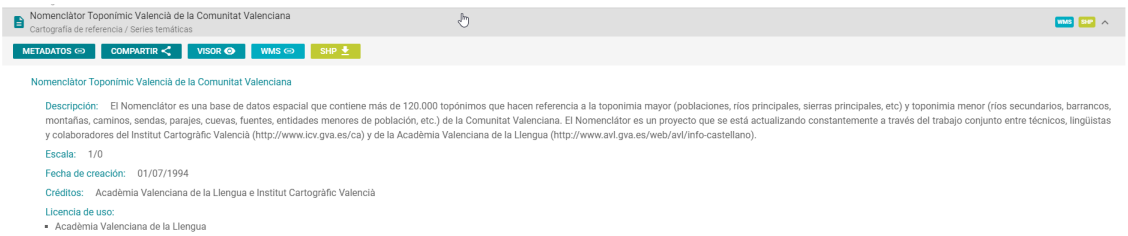


Ilustración 45.- Enlace al catálogo de la IDEV. Capa del "Nomenclátor".

La función que lo invoca, utiliza el complemento de *Python webbrowser* para la apertura en el navegador de la información de la capa de "*Nomenclator"* en la que se puede consultar el metadato, la url de conexión a los servicios que lo sirven y el enlace a la descarga de la base cartográfica en formato SHP.

7. Creación de un repositorio en GitHub.

Se ha creado una cuenta en GitHub para compartir el código fuente y subirlo a esta plataforma para que la comunidad de usuarios pueda consultar e incluso mejorar el código fuente. La URL del perfil creado es:

https://github.com/AlfonsoMoyaFuero/

Para descargar el complemento programado, se tiene que utilizar la siguiente dirección, que da acceso a todas las funcionalidades de GitHub:

https://github.com/AlfonsoMoyaFuero/idev_visor

Dentro del perfil que he generado, se crea un nuevo repositorio como se ve en la figura siguiente, dentro del portal de GitHub. En él se mantiene un control de versiones que permite a los usuarios descargar el proyecto en la fase deseada. Cada subida nueva o "*commit*" va documentada con los cambios producidos.

Otra de las ventajas es que se puede evolucionar e producto, creando diferentes ramas o "*branches*" del mismo, por si algún usuario quisiera aplicar esta programación a otras *IDEs*.

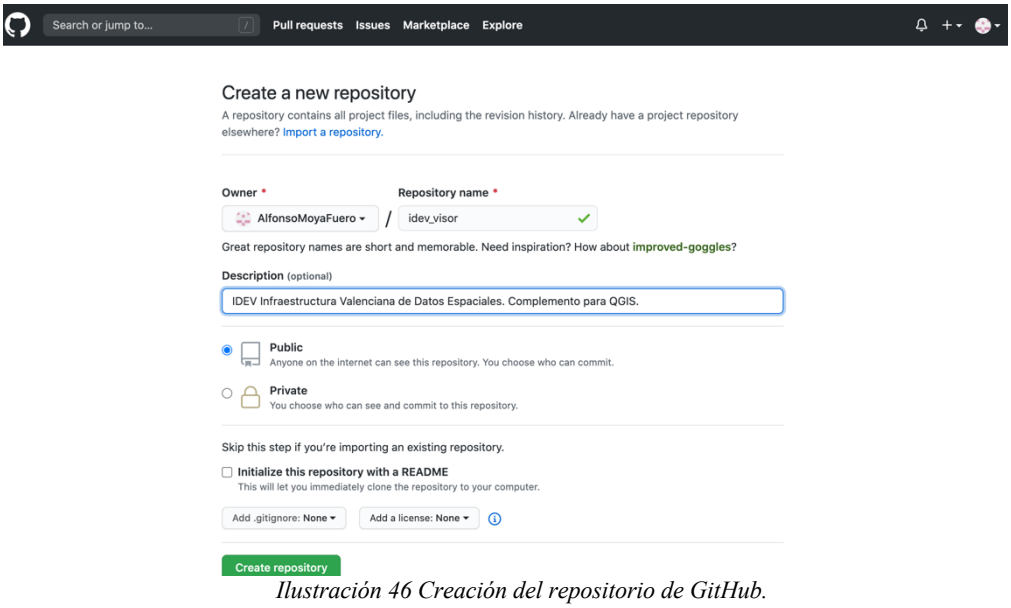


Ilustración 46 Creación del repositorio de GitHub.

Cuando se consulta el repositorio, hay un fichero información "*readme.m*d" que informa de las funcionalidades básicas del complemento y de la instalación del mismo. De igual manera, en los metadatos del complemento, se hace referencia a este repositorio para la consulta del código fuente.

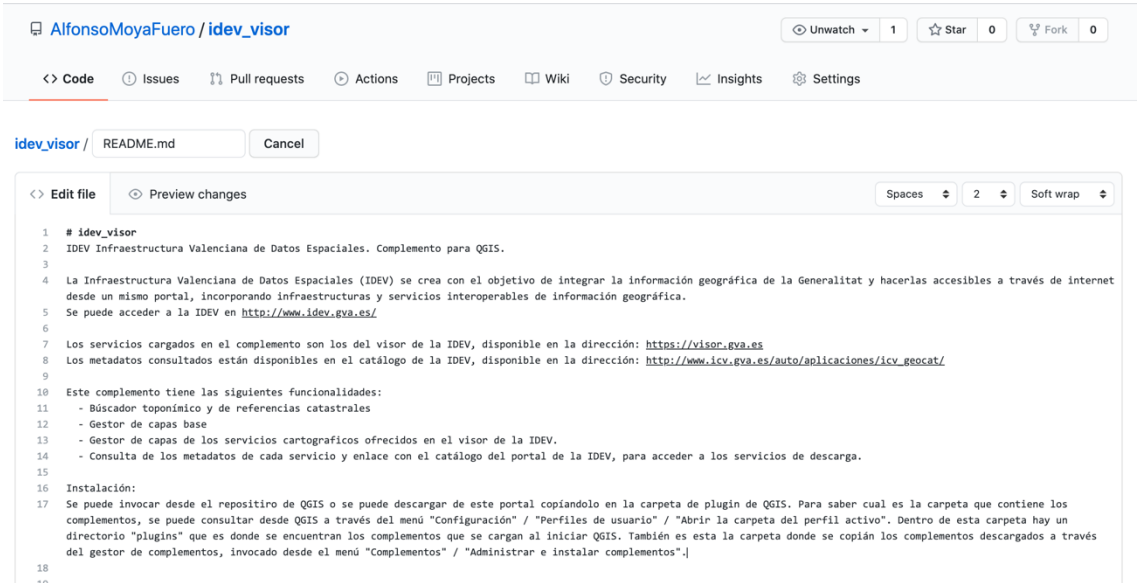


Ilustración 47 Fichero ·Readme.md" del repositorio del complemento.

Otra funcionalidad de GitHub es que cualquier usuario puede reportar errores de funcionamiento, abriendo un nuevo caso o "*issue*" en el que se describe el error. Todos los usuarios pueden contribuir en la resolución de los mismos. Cuando se crea el paquete para subirlo al repositorio oficial, uno de los campos del archivo "*metadata.txt*" que sirve para la descripción del mismo, es la *url* del "*tracker*" o web para referir errores. Más adelante veremos como se ha usado esta funcionalidad para corregir algunos aspectos que faltaban y que desde la propia comunidad de QGIS se ha indicado, ha sido necesario corregirlos para que se aprobara la publicación en producción del complemento. La dirección de esta página es:

https://github.com/AlfonsoMoyaFuero/idev_visor/issues

8. Publicación del complemento en el repositorio oficial de QGIS.

Para publicar el complemento desarrollado en el repositorio oficial de QGIS se necesita tener una cuenta oficial en OSGEO. Se ha solicitado la cuenta en dicho portal para poder subir el complemento. Se ha especificado el autor, el fin y la dirección del repositorio público de GitHub que alberga el código fuente.

Ilustración 48. - Inscripción en OsGeo para poder subir el complemento al repositorio oficial.

Se ha autorizado dicha cuenta con el nombre de usuario "*alfonsomoyafuero*". Una vez se tiene la cuenta activa, hay que logarse en la siguiente *url* que es la oficial de QGIS para poder subir complementos:

https://plugins.qgis.org/plugins/

Una vez en esta página y logados con el ususario de OsGeo hay que pulsar sobre el botón de "*Upload plugin*", se abre un diálogo para poder subir los ficheros necesarios y documentar el complemento.

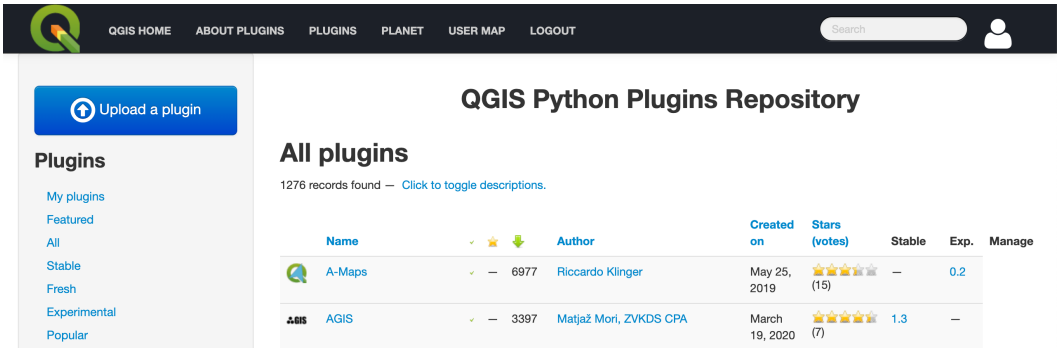


Ilustración 49. - Sitio oficial de QGIS de subida de complementos.

Para poder subir el complemento hay que comprimirlo en un paquete *ZIP* y utilizar el formulario web para poder subirlo.

Ilustración 50. - Formulario de subida del complemento en formato ZIP.

Una vez se sube este paquete hay que rellenar el resto de campos requeridos en un formulario. Si falta algún campo, el complemento se queda a la espera de subsanar los fallos. Hasta que no están corregidos no se aprueba el complemento y pasa a ser parte del repositorio oficial.

Para esa tarea de aprobación, es fundamenta la página antes descrita de "*issues*" que tiene el código fuente en *GitHub*. Es en esta página donde se hacen los comentarios necesarios para poder aprobar el complemento.

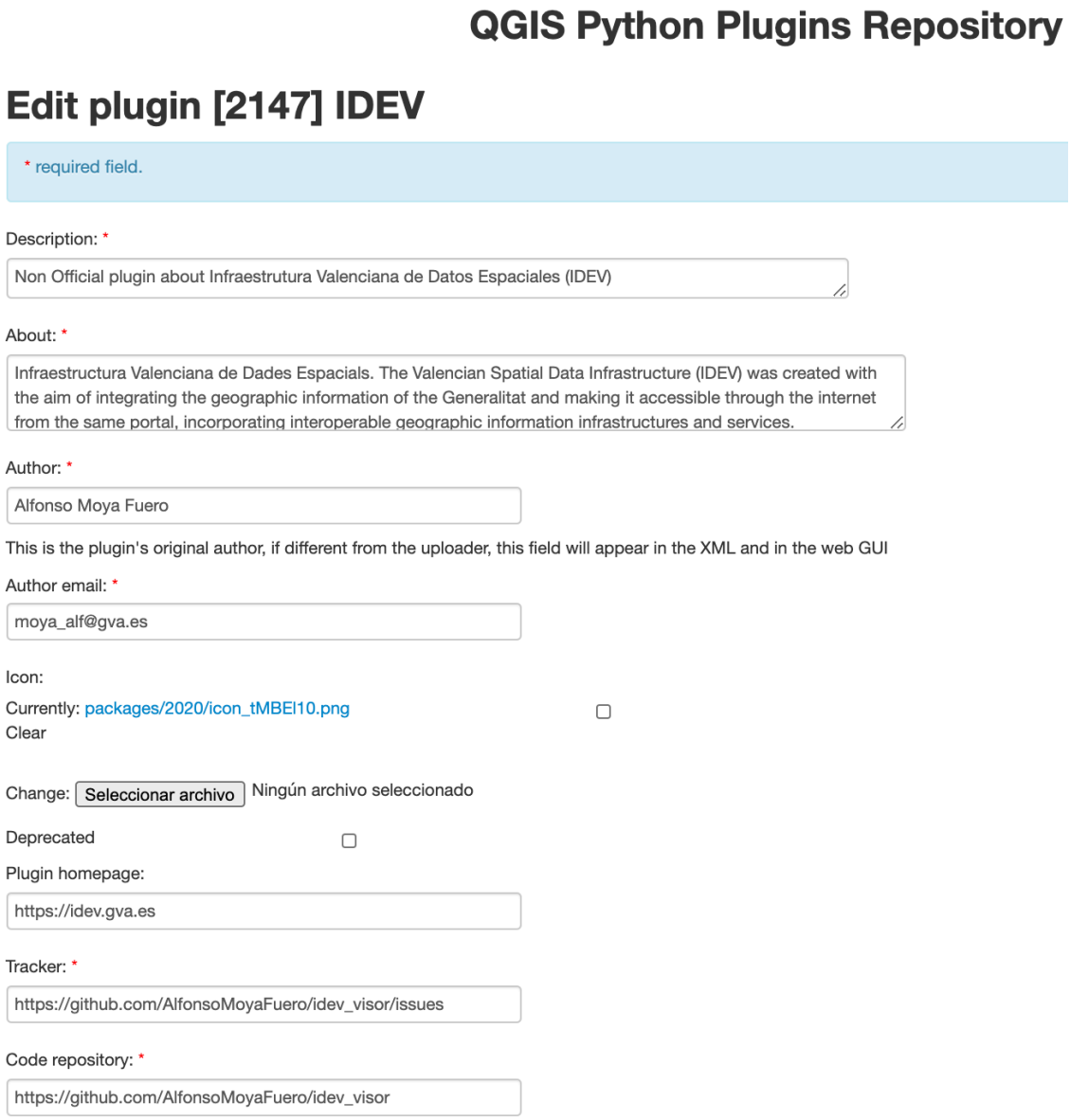


Ilustración 51. - Formulario de metadatos del complemento.

Las condiciones que tiene que cumplir el complemento para su aprobación son las siguientes, según se reporta en la ayuda de QGIS para desarrolladores, disponible en la url antes citada (https://plugins.qgis.org/publish/) y que son :

- El complemento debe ir documentado.
- En los metadatos se debe incluir los datos del creador, enlace al código fuente, especificaciones del tipo de licencia (al menos tipo *Creative Commons 2 GPL* o compatible) y página de rastreador o "*tracker*" para reportar errores.
- Respetar los componentes reutilizados.
- Si el componente usa dependencias eternas deben de ir referidas.
- Están prohibidos los ficheros binarios.

Estas recomendaciones son obligatorias aunque para mejorar la calidad de los complementos publicados se hacen más recomendaciones, de todas se especifican la que se han cumplido en el desarrollo de este complemento:

- Escribir los comentarios al código fuente en inglés.
- Colocar el componente en el menú adecuado y en una barra de herramientas.
- Comprobación de la no duplicidad del componente.
- Que el código fuente cargado en el *ZIP* utilizado para la publicación sea idéntico al código fuente del repositorio.
- Contener un archivo "*Readme*" y otro de "*License*".

Una vez subido el repositorio, en el el perfil del programador aparecen los componentes subidos y el estado de los mismos, incluyendo el número de descargas realizadas, puntuación de los usuarios, número de versión, fecha de subida…

Ilustración 52. - Información del complemento en el perfil de usuario.

A partir de esta interfaz, es donde se gestionan las versiones. En este caso es la versión 0.2 porque en la primera había algunas cosas que corregir y que se reportaron en la página del rastreador de *GitHub*. Una vez solucionados los errores y recomendaciones es cuando desde la comunidad de QGIS se le da el visto bueno (a través de usuarios con el rol de aprobadores) y pasa a estar disponible en el repositorio oficial.

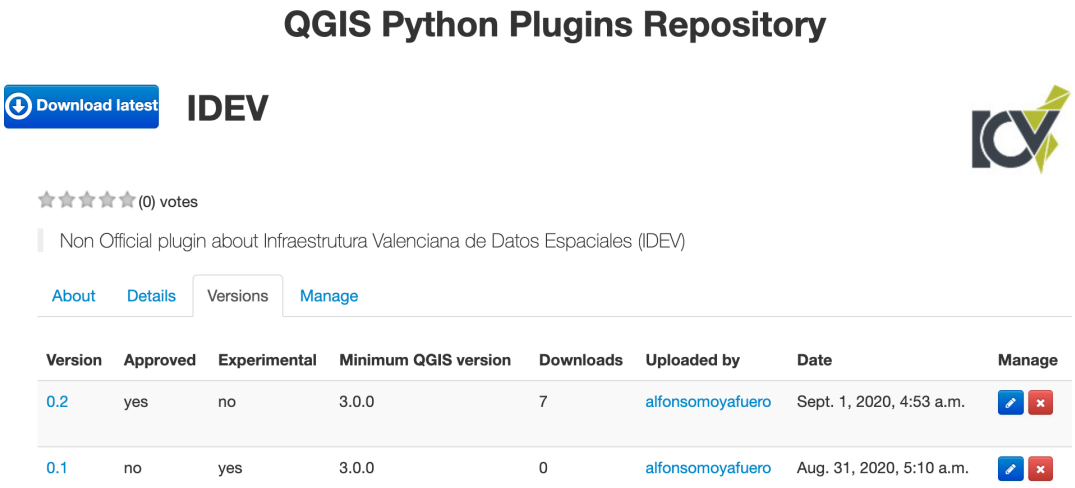


Ilustración 53. - Interfaz para el control de versiones del complemento.

Es también desde este portal de descarga de complementos donde se puede obtener la información de contacto y del *site* de rastreo para cualquier complemento. En este caso, esta sería la información pública del complemento:

Ilustración 54. - Información pública de contacto del complemento.

Con todas estas herramientas que se proporcionan se puede mantener una gestión de las versiones. A los usuarios que ya tengan instalado el complemento en su *QGIS* de escritorio, cuando se actualice a una nueva versión, les aparecerá un mensaje advirtiendo de que hay disponible una nueva versión, encargándose el gestor de complementos de descargar los nuevos ficheros.

9. Resultados obtenidos.

Se ha conseguido programar un complemento que cumple con los requerimientos de funcionalidad. El objetivo es insertar el gestor de capas del visor la IDEV dentro de QGIS. Entre los hitos conseguidos se destacan:

- Publicación en el repositorio oficial de QGIS del complemento, estando a disposición de la comunidad de usuarios. También está disponible el código fuente en el repositorio de GitHub, donde se pueden reportar errores y mejoras. Cualquier usuario puede evolucionarlo.
- Se ha conseguido replicar el funcionamiento del visor, con una interfaz similar y siguiendo la misma estructura de capas, para que, al usuario del portal, le sea familiar y amigable el uso del complemento dentro de QGIS.
- Se ha integrado dentro de QGIS el buscador toponímico de la IDEV, siendo una herramienta fundamental para la navegación por la cartografía ya la búsqueda se realiza sobre la siguiente información:
    - Nomenclátor oficial de la Academia Valenciana de la Lengua.
    - Direcciones postales y puntos kilométricos de la cartografía oficial del proyecto RT (Red de transportes).
    - Referencias catastrales de las parcelas de la Comunitat Valenciana.
- Integración del árbol de capas del visor de la IDEV dentro de QGIS replicando la misma estructura de padres/hijos.
- Buscador de capas de la IDEV para facilitar la activación de los servicios cartográficos.

- Elaboración de una memoria que describe paso a paso la elaboración del complemento, desde la creación de su esqueleto, su programación y subida a un repositorio público para que pueda ser reutilizado por los usuarios.
- Enlace de las capas del árbol con el catálogo de la IDEV a través de la interfaz del catálogo. Se accede para cada capa al metadato, a la dirección de los servicios en que se publica y a los servicios de descarga (en el caso de que esa capa de distribuya). Por ejemplo, las capas de la serie cartográfica BCV05 del ICV se pueden descargar por hojas, municipio o recorte personalizado en los formatos GEOJOSON, KML, PDF y SHP.

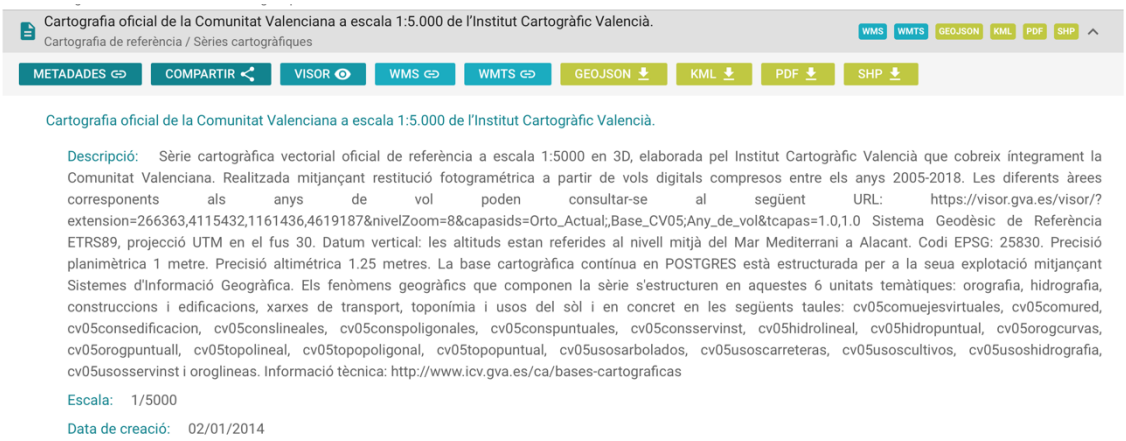


Ilustración 55. - Enlace con el catálogo de la IDEV para la serie BCV05.

Como colofón al resultado obtenido, exponemos un ejemplo de funcionalidad del complemento integrado dentro de un proyecto de SIG y que justifica que el desarrollo realizado no se limita a replicar la funcionalidad de un cliente ligero en *QGIS,* sino que permite incrementar la funcionalidad a través de las propias capacidades del SIG (ofrecidas por la *GUI* o programables utilizando la *API)* con la consecuente mejora de la puesta en valor de la información geográfica de la Comunitat Valenciana..

Dentro de los trabajos propios del Urbanismo y de la Ordenación del Territorio, es frecuente la tramitación de licencias para alguna actividad en concreto. Supongamos que una protectora de animales desea implantar un centro de recogida de mascotas y necesita solicitar los pertinentes permisos para poder iniciar la construcción y la actividad. Antes de iniciar todo el proceso, debe de asegurarse que la parcela en cuestión cumple con todos los requisitos que la legislación estipula para poder construir. Para ello es necesario consultar una serie de cartografías que marcan los límites a la ocupación del suelo. Dentro de la Comunitat Valenciana, se han recopilado todas las temáticas cartográficas que suponen un grado de protección del territorio, dentro de la denominada "*Infraestructura verde*". Estas capas delimitan el territorio según áreas protegidas (Parques Naturales, LICs, ZEPAs, Terreno Forestal Estratégico, Red Natura, Humedales Ramsar, …), áreas vulnerables (contaminación de acuíferos, zonas erosionables, …) y áreas con peligrosidad (zonas de riesgo de inundación, deslizamiento de laderas, …).

Todo este conjunto de capas limita las actividades que se pueden autorizar sobre los mismos, por lo que antes de solicitar las correspondientes licencias hay que consultar, para la parcela en la que se quiere ubicar la protectora, si está sujeta a algún tipo de restricción territorial que impida la concesión de la misma. Podemos destacar las siguientes limitaciones:

- Zona protegida. - Por ejemplo, si está dentro de un Parque Natural deberemos consultar el *PORN* y el *PRUG* del mismo para ver si está permitida dicho uso.
- Zona vulnerable. - Por ejemplo, puede estar dentro de una zona de contaminación de acuíferos, por lo que el proyecto deberá contener un plan de depuración de las aguas utilizadas.
- Zona peligrosa. - Por ejemplo, puede estar dentro de una zona de riesgo de inundación, por lo que impediría su instalación sin unas medidas correctoras, o incluso, si el riesgo es muy elevado, prohibiendo su implantación.
- Otra cuarta categoría la podemos establecer consultando el planeamiento urbanístico del municipio, se debe consultar la calificación para ver si está en suelo urbanizable, y la calificación, para ver si ese uso lo permite las ordenanzas para esa zona.

En resumen, antes de iniciar cualquier actividad sobre el territorio, hay que consultar innumerables capas de las cartografías oficiales que condicionan las concesiones de las pertinentes licencias. Se hace por ello imprescindible, disponer de una herramienta que, de una forma fácil e integrada dentro de nuestro programa de SIG, nos facilite la tarea de la carga y descarga de dichas capas. Consultar esta información a priori supone un ahorro de costes materiales y temporales porque nos informa primeramente de la viabilidad del proyecto, o también, en función del tipo de suelo en que se quiere implantar, de la petición a realizar a la administración. Por ejemplo, si está en suelo no urbanizable común, no se puede solicitar la licencia al no estar permitida la construcción, pero se puede invocar la figura del DIC (Declaración de Interés Comunitario) que sí permite para ciertos supuestos ocupar el territorio. Disponiendo de esta información a priori, ya se enfocaría toda la tramitación de los permisos hacia la figura que más probabilidad de otorgamiento tenga, ahorrando dilaciones innecesarias.

10. Conclusiones.

A la vista de los resultados obtenidos en el presente trabajo final de máster podemos concluir lo siguiente:

- Se ha conseguido integrar dentro de un software de SIG los servicios cartográficos de una IDE como es la IDEV de la Comunitat Valenciana, creando un complemento con las mismas funcionalidades que en el visor oficial de dicha infraestructura de datos.
- Poder disponer da los servicios cartográficos oficiales dentro del mismo software de SIG utilizado, facilita la elaboración de los proyectos de Urbanismo, Ordenación del Territorio y Paisaje, ya que pone a disposición del usuario toda la información necesaria de forma inmediata, sin necesidad de tener que acudir a fuentes externas o utilizar varios entornos de trabajo.
- Generar cartografías y servicios cartográficos supone un gran esfuerzo para las administraciones. La razón de ser de las IDEs es la compartición de dicha información de forma interoperable, estandarizada y sin restricciones. Este complemento ayuda a cumplir este fin ya que enlaza directamente con los productores de los datos, cargando de esta manera la información más actualizada.
- El complemento, además de cargar la información geográfica en el proyecto de QGIS por medio de un servicio de visualización, enlaza con el catálogo de la IDEV. Para cada capa catalogada, se muestra el metadato correspondiente, las url

de conexión de los servicios disponibles y enlaza con los servicios de descarga del archivo cartográfico fuente.

Como mejora al complemento, se contemplan las siguientes actuaciones:

- Es recomendable la traducción completa al inglés, no únicamente sus metadatos como está ahora, ya que esto aumentaría la cantidad de usuarios y mejoraría su accesibilidad.
- Se pueden añadir otras funcionalidades implementadas en el visor de la IDEV como son las consultas personalizadas. Desde el visor, por medio de unos desplegables, se invocan consultas predeterminadas en diferentes materias (aguas, arquitectura, biodiversidad, catastro, caza y pesca, espacios protegidos, infraestructuras viarias, …) siendo estas un complemento a la navegación por la cartografía junto con el buscador toponímico.
- Se ha informado al *Institut Cartogràfic Valencià* del complememto poniendo a disposición del mismo el código fuente por si se quisiera adoptar como un producto oficial del mismo, para poder publicitarlo desde su plataforma, añadirle sus logos, lo pudiera extender y darle carácter de oficialidad. Se subiría al repositorio oficial de QGIS mediante una cuenta corporativa en vez de la de un usuario particular. Esto aumentaría las funcionalidades y el número de usuarios. También es recomendable que el organismo lo adopte ya que la IDEV es una plataforma viva a la que se van añadiendo nuevos servicios. Mantener actualizadas las capas que se ofrecen desde el complemento es más sencillo desde el propio organismo ya que puede incluir es sus procedimientos de actualización del visor, la del complemento, evitando disparidades entre las capas ofrecidas desde ambos sitios.

11. Bibliografía.

Canadian Web Services (2020). Complemento de QGIS. (2020). Canadian Geospatial Data Infrastructure (CGDI). Recuperado en junio de 2020. https://www.nrcan.gc.ca/

Capdevila i Subirana, J. (2004). Infraestructura de datos espaciales (IDE). Definición y desarrollo actual en España. Scripta Nova. Revista Electrónica de Geografía y Ciencias Sociales. Universidad de Barcelona, Vol. VIII, núm. 170 (61).

Estado, LISIGE (2010). LEY 14/2010, de 5 de julio, sobre las infraestructuras y los servicios de información geográfica en España.

Geoportal IDESC (2020). (2020). Complemento de QGIS. Infraestructura de Datos Espaciales de Santiago de Cali (IDESC). Recuperado en junio de 2020. https://www.cali.gov.co/planeacion/publicaciones/3560/idesc/

IDE de Canarias. (2020). Infraestructura de Datos Espaciales de Canarias. Recuperado en junio de 2020 de https://www.idecanarias.es/

IDEE Infraestructura de Datos Espaciales de España (junio de 2020). Recuperado el 1 de junio de 2020 de http://www.idee.es

IEEE Institute of Electrical and Electronics Engineers. IEEE Standard Computer Dictionary: A Compilation of IEEE Standard Computer Glossaries. New York, NY: 1990.

Iniesto, M., & Núñez, A. (2014). Introducción a las infraestructuras de datos espaciales. M. Fomento, Madrid, España. Pág 22 a 26.

INSPIRE (2007). Directiva 2007/2/CE del Parlamento Europeo y del Consejo de 14 de marzo de 2007 por la que se establece una infraestructura de información espacial en la Comunidad Europea (INSPIRE).

French Addres (2020). Complemento de QGIS. Acceso al búscador de direcciones postales y toponímicas del gobierno francés. Recuperado en junio de 2020. https://geo.api.gouv.fr/adresse

ISO TC/211 Geographic technology standard models & schemas. (2002). Recuperado en junio de 2020 de https://www.isotc211.org

ISO 19115:2003 19119:2015 Metadata. (2015). Recuperado en junio de 2020 de https://www.iso.org/standard/53798.html

ISO 19128:2005 Geographic information — Web map server interface. (2005). Recuperado en junio de 2020 de https://www.iso.org/standard/32546.html

ISO 19142:2010 Geographic information — Web Feature Service (2010). Recuperado en junio de 2020 de https://www.iso.org/standard/42136.html

MTOPOOpenData (2020). Complemento de QGIS. Acceso a servicios webs del Ministerio de Transporte y Obras Públicas (MTOP) de Uruguay. Recuperado en junio de 2020. https://geoportal.mtop.gub.uy

OGC Open Geospatial Consortium (2020). Recuperado en junio de 2020 de https://www.ogc.org

QuickMapServices. (2020). Complemento de QGIS. *"Collection of easy to add basemaps*" Recuperado en junio de 2020. http://qms.nextgis.com

Rodríguez Pascual, A. F. (2008). El Proyecto IDEE y el SIG: contactos y sinergias. II Jornadas de SIG. Girona, (2008).

Solr (2020) Buscador e indexador basado en Apache Lucene de software libre. Recuprado de la web en agosto de 2020. https://lucene.apache.org/solr/

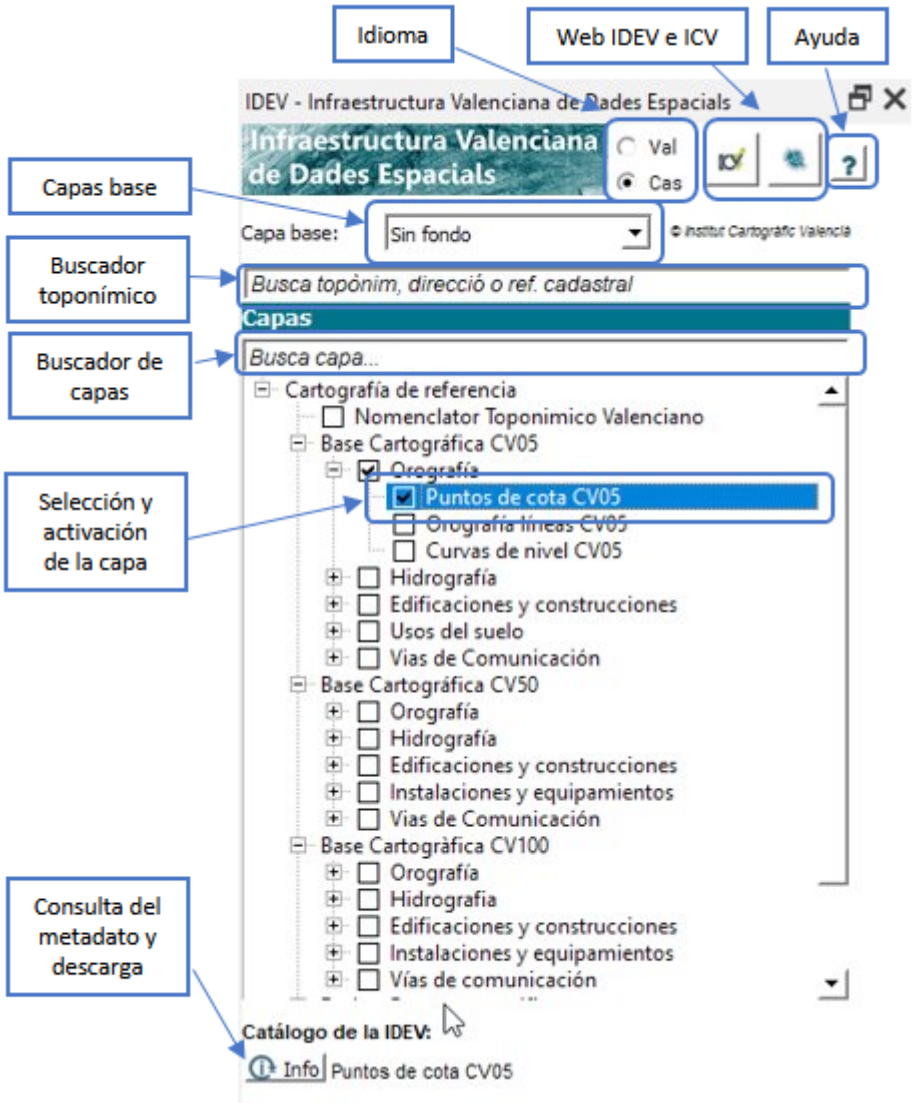
Spanish Inspire Catastral Downloader (2020). Complemento de QGIS. Descarga de cartografía catastral según Inspire. Recuperado en junio de 2020. http://www.catastro.minhap.gob.es/webinspire/index.html

Anexo I. Ayuda del complemento.

Manual de usuario:

El complemento permite visualizar y explotar diferentes capas de información geográfica provenientes de la IDEV. La interfaz permite al usuario hacer una búsqueda toponímica, búsqueda de capas, carga y descarga de servicios cartográficos y consulta del catálogo de la IDEV para cada capa.

Consta de un panel principal desde el que se acceden a todas las funcionalidades:

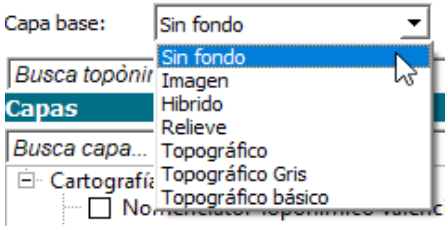


Selección del idioma: A través de dos "*radio buttons*" se escoge el idioma de la interfaz, pudiendo ser castellano o valenciano. Al seleccionar uno u otro se traduce toda la interfaz.

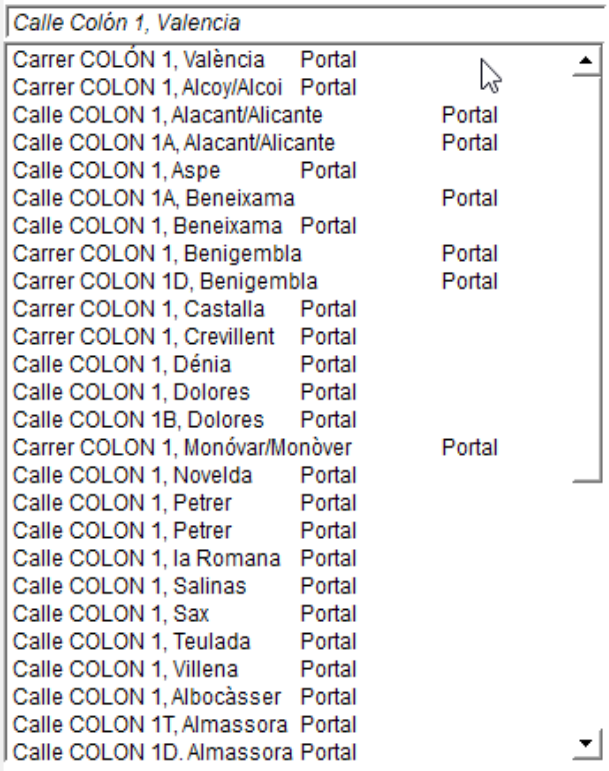
Enlace a la web de la IDEV y del ICV: A través de estos dos botones se accede la web de la "*Infraestructura Valenciana de Datos Espaciales"* y del "*Institut Cartogràfic Valencià".*

Ayuda: A través de este botón se abre el documento de ayuda en el navegador.

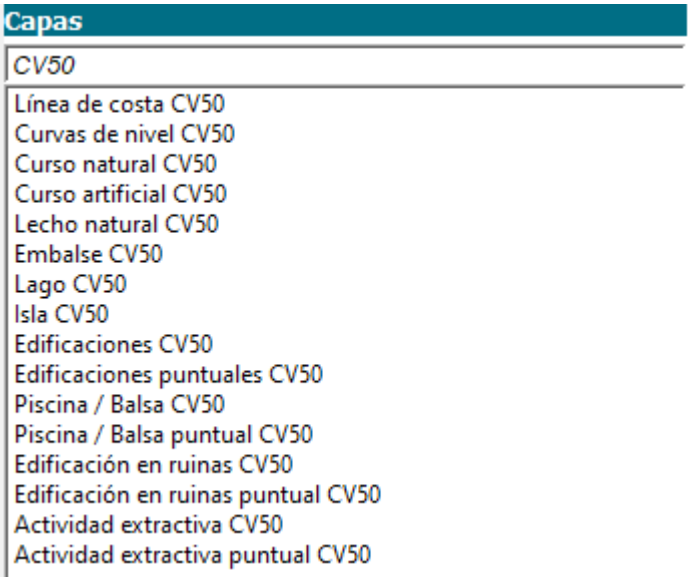
Selección de capas base: A través del desplegable de "*Capa base*" se selecciona la capa base que se quiera insertar en el proyecto de QGIS. Las capas seleccionadas se cargarán al final de la tabla de contenidos en el panel de "*Capas*".



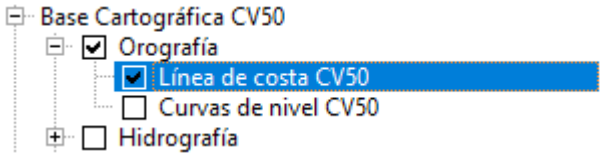
Buscador toponímico: Para poder buscar por texto libre un topónimo, una dirección postal o una referencia catastral. Por debajo del texto de entrada se muestra una lista con los resultados a medida que el usuario escribe el texto de búsqueda (se activa a partir de teclear el tercer carácter). Haciendo clic sobre cualquier resultado del listado, se centra el encuadre en la zona del topónimo, parcela catastral o dirección postal.



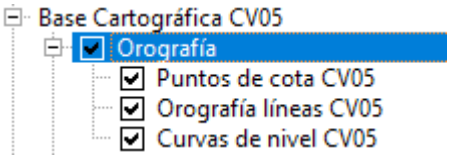
Buscador de capas: En este cuadro de texto se introduce el patrón de búsqueda para localizar una capa (por su nombre) dentro de la estructura de árbol. Se muestra en un listado las posibles coincidencias y pulsando sobre la capa que se quiera localizar, esta se muestra en su localización dentro del árbol con un sombreado gris.



Selección y activación de las capas: Podemos cargar una capa en el proyecto de QGIS activando el "*checkbox*" que hay al lado del nombre de la misma. Una vez activado se cargará con el mismo nombre que aparece en el árbol en la parte superior de la tabla de contenidos en el panel de "*Capas*". Para descargar la capa es tan sencillo como desactivar el "*checkbox*", automáticamente se descargará de la tabla de contenidos.

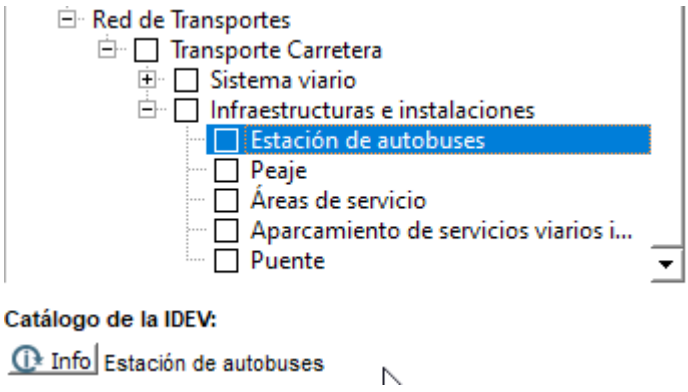


Se pueden activar o desactivar varias capas a la vez pulsando sobre el nodo padre que contiene un grupo de capas, en la siguiente figura se aprecia como pulsando sobre el nodo "Orografía" se activan las tres capas que cuelgan de dicho nodo.



De igual manera, pulsando sobre el nombre de una capa, vemos como se selecciona (es la capa que tiene el foco y se representa con un contorno azul por debajo). Es necesario que una capa está seleccionada para poder consultar su metadato o información de descarga.

Catálogo de la IDEV: Una vez se ha seleccionado una capa del árbol, se activa el botón de "*Info*" y se informa al lado del mismo del nombre de la capa que puede ser consultada su información. En la figura vemos como está seleccionada la capa de "Estación de autobuses" y como es posible enlazar con el catálogo de la IDEV pulsando sobre "*Info*":



Para poder consultar dicha información no es necesario que la capa esté cargada en el proyecto, únicamente se requiere que esté seleccionada. Se nos presenta en el navegador la interfaz del catálogo de la IDEV con la información referente a dicha capa.

Anexo II. Código fuente del complemento.

Como se indica en el apartado 7 de la memoria, todo el código fuente se encuentra disponible en el repositorio de GitHub. Se adjunta a continuación el código elaborado (se ha eliminado aquel código que automáticamente genera la herramienta "*Plugin builder*" y que forma parte de la estructura básica de todos los complementos).

Código fuente del fichero "*ide_visor_dockwidget.py*".

```
# -*- coding: utf-8 -*-
"""
/***************************************************************************
 IdevDockWidget
                                 A QGIS plugin
 Infraestructura Valenciana de Dades Espacials
 Generated by Plugin Builder: http://g-sherman.github.io/Qgis-Plugin-
Builder/
                             -------------------
        begin                : 2020-08-13
        git sha              : $Format:%H$
        copyright            : (C) 2020 by Alfonso Moya Fuero
        email                : moya_alf@gva.es

 ***************************************************************************
 *****/

/***************************************************************************
 *
 *
 *   This program is free software; you can redistribute it and/or
modify  *
 *   it under the terms of the GNU General Public License as published
by  *
 *   the Free Software Foundation; either version 2 of the License, or
 *
 *   (at your option) any later version.
 *
 *
 *

 ***************************************************************************
 *****/
"""

import os
import sys
import webbrowser
from PyQt5 import Qt, uic, QtGui, QtCore
from PyQt5.QtWidgets import QTreeWidgetItem, QMessageBox
from qgis.PyQt import *
from qgis.PyQt.QtCore import *
from qgis.utils import iface
from qgis.core import QgsRectangle, QgsRasterLayer, QgsProject,
QgsVectorLayer, QgsLayerTreeGroup, QgsLayerTreeLayer, \
    QgsLayerTreeNode, QgsGeometry, QgsPointXY, QgsPoint
from PyQt5.QtGui import QPixmap
```

```
import requests, json
from json import loads
from qgis.PyQt.QtCore import QUrl
from qgis.PyQt.QtWebKitWidgets import QWebView

FORM_CLASS, _ = uic.loadUiType(os.path.join(
    os.path.dirname(__file__), 'idev_visor_dockwidget_base.ui'))


class IdevDockWidget(QtWidgets.QDockWidget, FORM_CLASS):
    closingPlugin = pyqtSignal()

    def __init__(self, parent=None):
        """Constructor."""
        super(IdevDockWidget, self).__init__(parent)
        # Set up the user interface from Designer.
        # After setupUI you can access any designer object by doing
        # self.<objectname>, and you can use autoconnect slots - see
        # http://doc.qt.io/qt-5/designer-using-a-ui-file.html
        # #widgets-and-dialogs-with-auto-connect

        self.iface = iface  # Loading the iface class to control QGIS
events

        """ Configuration of the API event connections """
        self.setupUi(self)
        self.setupPlugin()  # Appearance initialization
        self.canvas = iface.mapCanvas()  # We take control of the
canvas through the panel
        self.list_toponimia.hide()  # Hide the ListWidget of the
results of the toponimic search
        self.list_capa.hide()  # We hide the list of the layer search
engine, it will appear when doing search

        """ Layer QTreeWidget Settings  """

        self.tree_toc.headerItem().setText(0, "")

        """ We dynamically load the layers in the QtreeWidget at
various parent / child levels """

        self.createTree()

        """ Dashboard widget event connectors """

self.rdb_cas.toggled.connect(lambda:self.changeLanguage(self.rdb_cas))
        self.rdb_val.toggled.connect(lambda:
self.changeLanguage(self.rdb_val))
        self.txt_toponimia.textChanged.connect(self.searchToponym)
        self.list_toponimia.itemClicked.connect(self.zoomToponym)

self.cmb_capas_base.currentIndexChanged.connect(self.changeBaseMap)
        self.tree_toc.itemClicked.connect(self.selectTocLayer)
        self.tree_toc.itemChanged.connect(self.checkTreeLayer)
        self.txt_busca_capa.textChanged.connect(self.searchLayer)
        self.list_capa.itemClicked.connect(self.selectTreeLayer)
        self.btn_meta.clicked.connect(self.openCatalog)
        self.btn_icv.clicked.connect(self.openIcvSite)
        self.btn_idev.clicked.connect(self.openIdevSite)
        self.btn_help.clicked.connect(self.openHelp)
```

```
    """ Function to load the initial appearance of the plugin  """
    def setupPlugin(self):
        baseDirectory = os.path.dirname(os.path.realpath(__file__)) #
Plugin absolute path
        self.im_icv =
QPixmap(os.path.dirname(os.path.realpath(__file__)) + '/IDEV.png')
        self.txt_logo_2.setPixmap(self.im_icv)    # Loading the IDEV
logo
        self.rdb_cas.setChecked(True)    # Default language
"Castellano"

self.btn_meta.setIcon(QtGui.QIcon(os.path.dirname(os.path.realpath(__f
ile__)) + '/info_on.png'))

self.btn_icv.setIcon(QtGui.QIcon(os.path.dirname(os.path.realpath(__fi
le__)) + '/icon.png'))

self.btn_idev.setIcon(QtGui.QIcon(os.path.dirname(os.path.realpath(__f
ile__)) + '/icon_idev.png'))

self.btn_help.setIcon(QtGui.QIcon(os.path.dirname(os.path.realpath(__f
ile__)) + '/help.png'))

    """ Function that changes the language of the controls. The
languages are Spanish / Valencian """

    def changeLanguage(self, b):
        self.createTree()  # Function that changes the language of the
controls. The languages are Spanish / Valencian
        self.txt_capa_seleccionada.setEnabled(False)  # Activate the
label of the active layer
        self.txt_capa_seleccionada.setText("")  # Claer the text of
the label layer name
        self.txt_info_capa.setEnabled(False)  # Desactivate the label
of the layer name
        self.btn_meta.setEnabled(False)  # Desactivate the button of
the info layer
        """ Check the language and set the text of the widgets """
        if b.text() == "Cas":
            if self.rdb_cas.isChecked():
                self.txt_toponimia.setText("Busca topónimo, dirección
o ref. catastral")
                self.txt_busca_capa.setText("Busca capa...")
                self.cmb_capas_base.setItemText(0, "Sin fondo")
                self.cmb_capas_base.setItemText(1, "Imagen")
                self.cmb_capas_base.setItemText(2, "Híbrido")
                self.cmb_capas_base.setItemText(3, "Relieve")
                self.cmb_capas_base.setItemText(4, "Topográfico")
                self.cmb_capas_base.setItemText(5, "Topográfico gris")
                self.cmb_capas_base.setItemText(6, "Topográfico
básico")
                self.txt_info_capa.setText("Catálogo de la IDEV:")
                """ Tooltilps translations to Spanish """
                self.btn_icv.setToolTip("Ir a la web del Institut
Cartogràfic Valencià")
                self.btn_idev.setToolTip("Ir a la web de la
Infraestructura Valenciana de Datos Espaciales")
                self.cmb_capas_base.setToolTip("Seleccionar capa base
de fondo")
                self.txt_toponimia.setToolTip("Inserte el texto para
la búsqueda toponímica")
```

```
                self.btn_meta.setToolTip("Pulsar para consultar el
catálogo de la IDEV de la capa seleccionada")
                self.rdb_cas.setToolTip("Activar para traducir el
complemento al castellano")
                self.rdb_val.setToolTip("Activar para traducir el
complemento al valenciano")
                self.txt_busca_capa.setToolTip("Inserte el texto para
la búsqueda de capas")
        if b.text() == "Val":
            if self.rdb_val.isChecked():
                self.txt_toponimia.setText("Busca topònim, direcció o
ref. cadastral")
                self.txt_busca_capa.setText("Busca capa...")
                self.cmb_capas_base.setItemText(0, "Sense fons")
                self.cmb_capas_base.setItemText(1, "Imatge")
                self.cmb_capas_base.setItemText(2, "Híbrid")
                self.cmb_capas_base.setItemText(3, "Relleu")
                self.cmb_capas_base.setItemText(4, "Topogràfic")
                self.cmb_capas_base.setItemText(5, "Topogràfic gris")
                self.cmb_capas_base.setItemText(6, "Topogràfic bàsic")
                self.txt_info_capa.setText("Catàleg de la IDEV:")
                """ Tooltilps translations to Valencian """
                self.btn_icv.setToolTip("Anar a la web de l'Institut
Cartogràfic Valencià")
                self.btn_idev.setToolTip("Anar a la web de la
Infraestructura Valenciana de Dades Espacials")
                self.cmb_capas_base.setToolTip("Seleccionar capa base
de fons")
                self.txt_toponimia.setToolTip("Inserisca el text per a
la cerca toponímica")
                self.btn_meta.setToolTip("Prémer per a consultar el
catàleg de la IDEV de la capa seleccionada")
                self.rdb_cas.setToolTip("Activar per a traduir el
complement al castellà")
                self.rdb_val.setToolTip("Activar per a traduir el
complement al valencià")
                self.txt_busca_capa.setToolTip("Inserisca el text per
a la cerca de capes")

    """  Function for toponymy search. Connection with the IDEV search
engine service """

    def searchToponym(self):
        # Disable the search with the default help text of the control
        if self.txt_toponimia.text() == 'Busca topónimo, dirección o
ref. catastral' or self.txt_toponimia.text() == 'Busca topònim,
direcció o ref. cadastral':
            pass
        # When change the text of the textline activate the search
        else:
            text = self.txt_toponimia.text()
            self.list_toponimia.clear()
            if len(text) > 2:  # The search began when type 3 or more
characters
                self.list_toponimia.show()  # Show the textlist of the
results of the search
                url =
'http://descargas.icv.gva.es/server_api/buscador/solrclient.php?start=
0&limit=40&query=' + text   # Create the url search query
                s = requests.get(str(url)).text  # Request the query
                start = s.find("[") + len("[")
```

```
                end = s.find("]")
                substring = "[" + s[start:end] + "]" # Format of the
response. Create a JSON file
                # print(substring)
                y = json.loads(substring)  # Load the result as a JSON
file
                i = len(y)  # Calculate the lenght of the query result
                for x in range(0, i):   # Load in the ListWidget the
results of the query
                    titulo = y[x]["titulo"]
                    clasificacion = y[x]["clasificacion"]
                    descripcion = y[x]["descripcion"]
                    if titulo.find('\n') != -1:
                        titulo = titulo.replace('\n', '')  # remove
that character of some query results
                    if descripcion == '':
                        descripcion = ''
                    else:
                        descripcion = "\n" + descripcion  # Format the
result in the ListWidget
                    widgetText =
QtWidgets.QListWidgetItem('{0}'.format(titulo) + "\t " + clasificacion
+ descripcion)
                    self.list_toponimia.addItem(widgetText)  # Add
items to the ListWidget
            else:
                self.list_toponimia.hide()  # If the query is null,
hide the ListWidget

    """ Function that centers the zoom on the toponym selected from
the ListWidget  """

    def zoomToponym(self):
        texto = self.list_toponimia.currentItem().text()  # Capture
the exact name of the toponym
        text, tipo = texto.split("\t")  # Split the information of the
toponym
        self.txt_toponimia.setText(text)  # Write the toponymin the
labeltext of search
        text = text.replace('/', '')  # Format the toponym text
        url =
'http://descargas.icv.gva.es/server_api/buscador/solrclient.php?start=
0&limit=40&query=' + text  # Create the query with the exact toponym
        s = requests.get(str(url)).text
        start = s.find("[") + len("[")
        end = s.find("]")
        substring = "[" + s[start:end] + "]"  # Format of the
response. Create a JSON file
        y = json.loads(substring)  # Load the result as a JSON file
        extent = y[0]["boundingbox"]  # Get the boundingbox of the
toponym
        XMin, Ymin, XMax, Ymax = extent.split(",")  # Asign the
coordinates of the extent
        zoomRectangle = QgsRectangle(float(XMin), float(Ymin),
float(XMax), float(Ymax))  # Create a zoomRectangle
        self.iface.mapCanvas().setExtent(zoomRectangle)  # Set the
extent o the project
        if XMin == XMax:
            self.iface.mapCanvas().zoomScale(1000)  # Default scale if
the toponym is a puntual shape
        self.iface.mapCanvas().refresh()  # Refresh the canvas view
```

```
        self.list_toponimia.hide()  # Hide the listwidget

    """ Function to load the background base layers  """

    def changeBaseMap(self):
        extent = self.canvas.extent()  # Store the actual extent in a
variable
        names = [layer.name() for layer in
QgsProject.instance().mapLayers().values()]  # Store in a list the
names of the layers
        for i in names:  # Unload the layer by name
            if i == 'Contorno autonómico':
                self.unloadLayerToc('Contorno autonómico')
            if i == 'Ortofoto 2019':
                self.unloadLayerToc('Ortofoto 2019')
            if i == 'ESRI Satellite':
                self.unloadLayerToc('ESRI Satellite')
            if i == 'Capa base Híbrido':
                self.unloadLayerToc('Capa base Híbrido')
            if i == 'Capa base Relieve':
                self.unloadLayerToc('Capa base Relieve')
            if i == 'Capa base Topográfico':
                self.unloadLayerToc('Capa base Topográfico')
            if i == 'Capa base Topográfico Básico':
                self.unloadLayerToc('Capa base Topográfico Básico')
            if i == 'Capa base Topográfico Gris':
                self.unloadLayerToc('Capa base Topográfico Gris')

        """ Loading the different base layers in the project  """

        if self.cmb_capas_base.currentIndex() == 1:
            urlWithParams_cv =
"crs=EPSG:25830&dpiMode=7&format=image/png&layers=limites_continuo&sty
les&url=https://terramapas.icv.gva.es/cgi-
bin/mapserv.fcgi?map%3D/srv_apl/mapserv/servicios/01_cartografia/05_un
idadesadm/01_contorno_autonomico/contorno_autonomico.map"
            rasterLyr_cv = QgsRasterLayer(urlWithParams_cv, 'Contorno
autonómico', 'wms')
            if not rasterLyr_cv.isValid():
                print("Capa no válida")
                iface.messageBar().pushMessage("Error", "Capa no
válida", level=Qgis.Critical, duration=5)
            else:
                QgsProject.instance().addMapLayer(rasterLyr_cv, False)
                layerTree = iface.layerTreeCanvasBridge().rootGroup()
                layerTree.insertChildNode(-1,
QgsLayerTreeLayer(rasterLyr_cv))
            urlWithParams_orto =
'crs=EPSG:3857&dpiMode=7&format=image/png;%20mode%3D8bit&layers=01_8bi
ts_01_RGB_05_PNG&styles&tileMatrixSet=01_8bits_01_RGB_05_PNG-wmsc-
0&url=http://terramapas.icv.gva.es/odcv05_etrs89h30_2019_3857'
            rasterLyr_orto = QgsRasterLayer(urlWithParams_orto,
'Ortofoto 2019', 'wms')
            if not rasterLyr_orto.isValid():
                print("Capa no válida")
                iface.messageBar().pushMessage("Error", "Capa no
válida", level=Qgis.Critical, duration=5)
            else:
                QgsProject.instance().addMapLayer(rasterLyr_orto,
False)
                layerTree = iface.layerTreeCanvasBridge().rootGroup()
```

```
                layerTree.insertChildNode(-1,
QgsLayerTreeLayer(rasterLyr_orto))
            urlWithParams_esri =
'type=xyz&zmin=0&zmax=20&url=https://server.arcgisonline.com/ArcGIS/re
st/services/World_Imagery/MapServer/tile/{z}/{y}/{x}'
            rasterLyr_esri = QgsRasterLayer(urlWithParams_esri, 'ESRI
Satellite', 'wms')
            if not rasterLyr_esri.isValid():
                print("Capa no válida")
                iface.messageBar().pushMessage("Error", "Capa no
válida", level=Qgis.Critical, duration=5)
            else:
                QgsProject.instance().addMapLayer(rasterLyr_esri,
False)
                layerTree = iface.layerTreeCanvasBridge().rootGroup()
                layerTree.insertChildNode(-1,
QgsLayerTreeLayer(rasterLyr_esri))

        if self.cmb_capas_base.currentIndex() == 2:
            urlWithParams_hibrido =
'crs=EPSG:3857&dpiMode=7&format=image/png&layers=mapabase_hibrid&style
s&tileMatrixSet=mapabase_hibrid-wmsc-
0&url=http://terramapas.icv.gva.es/mapabase_hibrid/'
            rasterLyr_hibrido = QgsRasterLayer(urlWithParams_hibrido,
'Capa base Híbrido', 'wms')
            if not rasterLyr_hibrido.isValid():
                print("Capa no válida")
                iface.messageBar().pushMessage("Error", "Capa no
válida", level=Qgis.Critical, duration=5)
            else:
                QgsProject.instance().addMapLayer(rasterLyr_hibrido,
False)
                layerTree = iface.layerTreeCanvasBridge().rootGroup()
                layerTree.insertChildNode(-1,
QgsLayerTreeLayer(rasterLyr_hibrido))
            urlWithParams_orto =
'crs=EPSG:3857&dpiMode=7&format=image/png;%20mode%3D8bit&layers=01_8bi
ts_01_RGB_05_PNG&styles&tileMatrixSet=01_8bits_01_RGB_05_PNG-wmsc-
0&url=http://terramapas.icv.gva.es/odcv05_etrs89h30_2019_3857'
            rasterLyr_orto = QgsRasterLayer(urlWithParams_orto,
'Ortofoto 2019', 'wms')
            if not rasterLyr_orto.isValid():
                print("Capa no válida")
                iface.messageBar().pushMessage("Error", "Capa no
válida", level=Qgis.Critical, duration=5)
            else:
                QgsProject.instance().addMapLayer(rasterLyr_orto,
False)
                layerTree = iface.layerTreeCanvasBridge().rootGroup()
                layerTree.insertChildNode(-1,
QgsLayerTreeLayer(rasterLyr_orto))
            urlWithParams_esri =
'type=xyz&zmin=0&zmax=20&url=https://server.arcgisonline.com/ArcGIS/re
st/services/World_Imagery/MapServer/tile/{z}/{y}/{x}'
            rasterLyr_esri = QgsRasterLayer(urlWithParams_esri, 'ESRI
Satellite', 'wms')
            if not rasterLyr_esri.isValid():
                print("Capa no válida")
                iface.messageBar().pushMessage("Error", "Capa no
válida", level=Qgis.Critical, duration=5)
            else:
```

```
                QgsProject.instance().addMapLayer(rasterLyr_esri,
False)
                layerTree = iface.layerTreeCanvasBridge().rootGroup()
                layerTree.insertChildNode(-1,
QgsLayerTreeLayer(rasterLyr_esri))

        if self.cmb_capas_base.currentIndex() == 3:
            urlWithParams_cv =
'crs=EPSG:25830&dpiMode=7&format=image/png&layers=limites_continuo&sty
les&url=https://terramapas.icv.gva.es/cgi-
bin/mapserv.fcgi?map%3D/srv_apl/mapserv/servicios/01_cartografia/05_un
idadesadm/01_contorno_autonomico/contorno_autonomico.map'
            rasterLyr_cv = QgsRasterLayer(urlWithParams_cv, 'Contorno
autonómico', 'wms')
            if not rasterLyr_cv.isValid():
                print("Capa no válida")
                iface.messageBar().pushMessage("Error", "Capa no
válida", level=Qgis.Critical, duration=5)
            else:
                QgsProject.instance().addMapLayer(rasterLyr_cv, False)
                layerTree = iface.layerTreeCanvasBridge().rootGroup()
                layerTree.insertChildNode(-1,
QgsLayerTreeLayer(rasterLyr_cv))
            urlWithParams_relieve =
'crs=EPSG:3857&dpiMode=7&format=image/png&layers=01_8bits_01_RGB_05_PN
G&styles&tileMatrixSet=01_8bits_01_RGB_05_PNG-wmsc-
1&url=http://terramapas.icv.gva.es/mapabase_isohipsas'
            rasterLyr_relieve = QgsRasterLayer(urlWithParams_relieve,
'Capa base Relieve', 'wms')
            if not rasterLyr_relieve.isValid():
                print("Capa no válida")
                iface.messageBar().pushMessage("Error", "Capa no
válida", level=Qgis.Critical, duration=5)
            else:
                QgsProject.instance().addMapLayer(rasterLyr_relieve,
False)
                layerTree = iface.layerTreeCanvasBridge().rootGroup()
                layerTree.insertChildNode(-1,
QgsLayerTreeLayer(rasterLyr_relieve))

        if self.cmb_capas_base.currentIndex() == 4:
            urlWithParams_cv =
'crs=EPSG:25830&dpiMode=7&format=image/png&layers=limites_continuo&sty
les&url=https://terramapas.icv.gva.es/cgi-
bin/mapserv.fcgi?map%3D/srv_apl/mapserv/servicios/01_cartografia/05_un
idadesadm/01_contorno_autonomico/contorno_autonomico.map'
            rasterLyr_cv = QgsRasterLayer(urlWithParams_cv, 'Contorno
autonómico', 'wms')
            if not rasterLyr_cv.isValid():
                print("Capa no válida")
                iface.messageBar().pushMessage("Error", "Capa no
válida", level=Qgis.Critical, duration=5)
            else:
                QgsProject.instance().addMapLayer(rasterLyr_cv, False)
                layerTree = iface.layerTreeCanvasBridge().rootGroup()
                layerTree.insertChildNode(-1,
QgsLayerTreeLayer(rasterLyr_cv))
            urlWithParams_topografico =
'crs=EPSG:3857&dpiMode=7&format=image/png;%20mode%3D8bit&layers=topogr
afico_continuo_epsg3857&styles&tileMatrixSet=topografico_continuo_epsg
3857-wmsc-0&url=http://terramapas.icv.gva.es/mapabase_topografico/'
```

```
            rasterLyr_topografico =
QgsRasterLayer(urlWithParams_topografico, 'Capa base Topográfico',
'wms')
            if not rasterLyr_topografico.isValid():
                print("Capa no válida")
                iface.messageBar().pushMessage("Error", "Capa no
válida", level=Qgis.Critical, duration=5)
            else:

QgsProject.instance().addMapLayer(rasterLyr_topografico, False)
                layerTree = iface.layerTreeCanvasBridge().rootGroup()
                layerTree.insertChildNode(-1,
QgsLayerTreeLayer(rasterLyr_topografico))

        if self.cmb_capas_base.currentIndex() == 5:
            urlWithParams_cv =
'crs=EPSG:25830&dpiMode=7&format=image/png&layers=limites_continuo&sty
les&url=https://terramapas.icv.gva.es/cgi-
bin/mapserv.fcgi?map%3D/srv_apl/mapserv/servicios/01_cartografia/05_un
idadesadm/01_contorno_autonomico/contorno_autonomico.map'
            rasterLyr_cv = QgsRasterLayer(urlWithParams_cv, 'Contorno
autonómico', 'wms')
            if not rasterLyr_cv.isValid():
                print("Capa no válida")
                iface.messageBar().pushMessage("Error", "Capa no
válida", level=Qgis.Critical, duration=5)
            else:
                QgsProject.instance().addMapLayer(rasterLyr_cv, False)
                layerTree = iface.layerTreeCanvasBridge().rootGroup()
                layerTree.insertChildNode(-1,
QgsLayerTreeLayer(rasterLyr_cv))

            urlWithParams_gris =
'crs=EPSG:3857&dpiMode=7&format=image/png;%20mode%3D8bit&layers=mapaba
se_topografico_grises&styles&tileMatrixSet=mapabase_topografico_grises
-wmsc-0&url=http://terramapas.icv.gva.es/mapabase_topografico_grises'
            rasterLyr_gris = QgsRasterLayer(urlWithParams_gris, 'Capa
base Topográfico Gris', 'wms')
            if not rasterLyr_gris.isValid():
                print("Capa no válida")
                iface.messageBar().pushMessage("Error", "Capa no
válida", level=Qgis.Critical, duration=5)
            else:
                QgsProject.instance().addMapLayer(rasterLyr_gris,
False)
                layerTree = iface.layerTreeCanvasBridge().rootGroup()
                layerTree.insertChildNode(-1,
QgsLayerTreeLayer(rasterLyr_gris))

        if self.cmb_capas_base.currentIndex() == 6:
            urlWithParams_cv =
'crs=EPSG:25830&dpiMode=7&format=image/png&layers=limites_continuo&sty
les&url=https://terramapas.icv.gva.es/cgi-
bin/mapserv.fcgi?map%3D/srv_apl/mapserv/servicios/01_cartografia/05_un
idadesadm/01_contorno_autonomico/contorno_autonomico.map'
            rasterLyr_cv = QgsRasterLayer(urlWithParams_cv, 'Contorno
autonómico', 'wms')
            if not rasterLyr_cv.isValid():
                print("Capa no válida")
                iface.messageBar().pushMessage("Error", "Capa no
válida", level=Qgis.Critical, duration=5)
```

```
            else:
                QgsProject.instance().addMapLayer(rasterLyr_cv, False)
                layerTree = iface.layerTreeCanvasBridge().rootGroup()
                layerTree.insertChildNode(-1,
QgsLayerTreeLayer(rasterLyr_cv))
            urlWithParams_basico =
'crs=EPSG:3857&dpiMode=7&format=image/png&layers=mapabase_basico&style
s&tileMatrixSet=mapabase_basico-wmsc-
0&url=http://terramapas.icv.gva.es/mapabase_basico/'
            rasterLyr_basico = QgsRasterLayer(urlWithParams_basico,
'Capa base Topográfico Básico', 'wms')
            if not rasterLyr_basico.isValid():
                print("Capa no válida")
                iface.messageBar().pushMessage("Error", "Capa no
válida", level=Qgis.Critical, duration=5)
            else:
                QgsProject.instance().addMapLayer(rasterLyr_basico,
False)
                layerTree = iface.layerTreeCanvasBridge().rootGroup()
                layerTree.insertChildNode(-1,
QgsLayerTreeLayer(rasterLyr_basico))

        iface.mapCanvas().setExtent(extent)
        iface.mapCanvas().refresh()

    """ Selecting a layer in the layer tree and activating the catalog
button  """

    #@QtCore.pyqtSlot(QtWidgets.QTreeWidgetItem, int)
    def selectTocLayer(self):
        i = 0
        for ix in self.tree_toc.selectedIndexes():
            if self.rdb_cas.isChecked() == True:
                for nombre in capasIDEV:
                    if ix.data() == nombre['nombre_cas'] and i == 0:
                        self.btn_meta.setEnabled(True)
                        self.txt_info_capa.setEnabled(True)

self.txt_capa_seleccionada.setText(nombre['nombre_cas'])
                        self.txt_capa_seleccionada.setEnabled(True)
                        i = i  + 1

            if self.rdb_val.isChecked() == True:
                for nombre in capasIDEV:
                    if ix.data() == nombre['nombre_val']:
                        self.btn_meta.setEnabled(True)
                        self.txt_info_capa.setEnabled(True)

self.txt_capa_seleccionada.setText(nombre['nombre_val'])
                        self.txt_capa_seleccionada.setEnabled(True)

    """ Button to open the ICV website   """

    def openIcvSite(self):
        if self.rdb_cas.isChecked() == True:
            webbrowser.open('http://www.icv.gva.es/es/inicio')
        if self.rdb_val.isChecked() == True:
            webbrowser.open('http://www.icv.gva.es/va/inicio')

    """ Button to open the IDEV website   """
```

```
    def openIdevSite(self):
        if self.rdb_cas.isChecked() == True:
            webbrowser.open('http://www.idev.gva.es/es/inicio')
        if self.rdb_val.isChecked() == True:
            webbrowser.open('http://www.idev.gva.es/va/inicio')

    def openHelp(self):
        if self.rdb_cas.isChecked() == True:

webbrowser.open(os.path.dirname(os.path.realpath(__file__)) +
'/data/help_cast.html')
        if self.rdb_val.isChecked() == True:

webbrowser.open(os.path.dirname(os.path.realpath(__file__)) +
'/data/help_val.html')

    """ Button to open the catalog website of the selected layer in
the QTreeWidget """

    def openCatalog(self):  # The layer name is different in each
language, there is a different search in the layer dictionary
        if self.rdb_cas.isChecked() == True:
            for nombre in capasIDEV:
                if self.txt_capa_seleccionada.text() ==
nombre['nombre_cas']:

webbrowser.open('http://www.icv.gva.es/auto/aplicaciones/icv_geocat/#/
search?uuid=' + nombre['metadato']) # Open the catalog
        if self.rdb_val.isChecked() == True:
            for nombre in capasIDEV:
                if self.txt_capa_seleccionada.text() ==
nombre['nombre_val']:

webbrowser.open('http://www.icv.gva.es/auto/aplicaciones/icv_geocat/#/
search?uuid=' + nombre['metadato']) # Open the catalog

    """ Load the layer in the QGIS project whrn check the item in the
QTreeWidget """

    def checkTreeLayer(self, it, col): # The layer name is different
in each language, there is a different search in the layer dictionary
        if self.rdb_cas.isChecked() == True:
            for nombre in capasIDEV:  # Search in the layer dictionary
by spanish name
                if it.checkState(0) == QtCore.Qt.Checked:
                    if it.text(col) == nombre['nombre_cas']:
                        self.loadLayerToc(nombre['url'],
nombre['nombre_cas'])  # load the layer Checked in the QGIS project
                        if nombre['aviso_cas'] != "":
                            msg = QMessageBox.information(self,
"Aviso:", nombre['aviso_cas'])  # Show advice if is not null
                            #msg.exec()
                else:
                    if it.text(col) == nombre['nombre_cas']:
                        self.unloadLayerToc(nombre['nombre_cas']) #
Unload the layer of the proyect if is unchecked

        if self.rdb_val.isChecked() == True:
            for nombre in capasIDEV:  # Search in the layer dictionary
by valencian name
                if it.checkState(0) == QtCore.Qt.Checked:
```

```
                    if it.text(col) == nombre['nombre_val']:
                        self.loadLayerToc(nombre['url'],
nombre['nombre_val'])  # load the layer Checked in the QGIS project
                        if nombre['aviso_val'] != "":
                            msg = QMessageBox.information(self,
"Avís:", nombre['aviso_val']) # Show advice if is not null
                            #msg.exec()
                else:
                    if it.text(col) == nombre['nombre_val']:
                        self.unloadLayerToc(nombre['nombre_val'])  #
Unload the layer of the proyect if is unchecked

    """ Funcion for load a layer in the QGIS project by name and url
connection string """

    def loadLayerToc(self, url, nombre):
        layer = QgsRasterLayer(url, nombre, 'wms')  # Create a PyQt
layer
        if not layer.isValid():
            print("Capa no válida")
            iface.messageBar().pushMessage("Error", "Capa no válida",
level=Qgis.Critical, duration=5)
        else:
            QgsProject.instance().addMapLayer(layer, False)  # Add the
layer in the project
            layerTree = iface.layerTreeCanvasBridge().rootGroup()  #
Add the layer on the top of the TOC
            layerTree.insertChildNode(0, QgsLayerTreeLayer(layer))

    """ Funcion for unload a layer in the QGIS project by layers name
"""

    def unloadLayerToc(self, nombre):

QgsProject.instance().removeMapLayer(QgsProject.instance().mapLayersBy
Name(nombre)[0])  # Unload the layer form the TOC
        iface.mapCanvas().refresh()  # Refresh the projet view in QGIS

    """ Function to dynamically load the layers in the QTreeWidget in
the corresponding language """

    def createTree(self):
        self.tree_toc.clear() # Clear the tree
        carto = QTreeWidgetItem(self.tree_toc)  # Create the
QtreeWidget
        if self.rdb_cas.isChecked() == True: # Create the fahther and
child nodes in spanish
            carto.setText(0, "Cartografía de referencia") # First item
of the tree, father 0
            # carto.setFlags(carto.flags() | Qt.ItemIsTristate |
Qt.ItemIsUserCheckable)
            toponimia = QTreeWidgetItem(carto)
            toponimia.setFlags(toponimia.flags() |
Qt.ItemIsUserCheckable)
            toponimia.setText(0, "Nomenclator Toponimico Valenciano")
            toponimia.setCheckState(0, Qt.Unchecked)
            basecartografica05 = QTreeWidgetItem(carto)
            basecartografica05.setText(0, "Base Cartográfica CV05")
            orografia05 = QTreeWidgetItem(basecartografica05)
            orografia05.setFlags(orografia05.flags() |
Qt.ItemIsTristate | Qt.ItemIsUserCheckable)
```

```
            orografia05.setText(0, "Orografía")
            orografia05.setCheckState(0, Qt.Unchecked)
            puntoscota = QTreeWidgetItem(orografia05)
            puntoscota.setFlags(puntoscota.flags() |
Qt.ItemIsUserCheckable)
            puntoscota.setText(0, "Puntos de cota CV05")
            puntoscota.setCheckState(0, Qt.Unchecked)
            orolineas = QTreeWidgetItem(orografia05)
            orolineas.setFlags(orolineas.flags() |
Qt.ItemIsUserCheckable)
            orolineas.setText(0, "Orografía líneas CV05")
            orolineas.setCheckState(0, Qt.Unchecked)
            curvasnivel = QTreeWidgetItem(orografia05)
            curvasnivel.setFlags(orolineas.flags() |
Qt.ItemIsUserCheckable)
            curvasnivel.setText(0, "Curvas de nivel CV05")
            curvasnivel.setCheckState(0, Qt.Unchecked)
            hidrografia05 = QTreeWidgetItem(basecartografica05)
            hidrografia05.setFlags(hidrografia05.flags() |
Qt.ItemIsTristate | Qt.ItemIsUserCheckable)
            hidrografia05.setText(0, "Hidrografía")
            hidrografia05.setCheckState(0, Qt.Unchecked)
            hidropuntos05 = QTreeWidgetItem(hidrografia05)
            hidropuntos05.setFlags(orolineas.flags() |
Qt.ItemIsUserCheckable)
            hidropuntos05.setText(0, "Hidrografía puntual CV05")
            hidropuntos05.setCheckState(0, Qt.Unchecked)
            hidrolineas05 = QTreeWidgetItem(hidrografia05)
            hidrolineas05.setFlags(orolineas.flags() |
Qt.ItemIsUserCheckable)
            hidrolineas05.setText(0, "Hidrografía lineal CV05")
            hidrolineas05.setCheckState(0, Qt.Unchecked)
            construcciones05 = QTreeWidgetItem(basecartografica05)
            construcciones05.setFlags(construcciones05.flags() |
Qt.ItemIsTristate | Qt.ItemIsUserCheckable)
            construcciones05.setText(0, "Edificaciones y
construcciones")
            construcciones05.setCheckState(0, Qt.Unchecked)
            construpuntos05 = QTreeWidgetItem(construcciones05)
            construpuntos05.setFlags(construpuntos05.flags() |
Qt.ItemIsUserCheckable)
            construpuntos05.setText(0, "Construcciones puntuales
CV05")
            construpuntos05.setCheckState(0, Qt.Unchecked)
            construservi05 = QTreeWidgetItem(construcciones05)
            construservi05.setFlags(construservi05.flags() |
Qt.ItemIsUserCheckable)
            construservi05.setText(0, "Servicios e instalaciones
CV05")
            construservi05.setCheckState(0, Qt.Unchecked)
            construlinea05 = QTreeWidgetItem(construcciones05)
            construlinea05.setFlags(construlinea05.flags() |
Qt.ItemIsUserCheckable)
            construlinea05.setText(0, "Construcciones lineales CV05")
            construlinea05.setCheckState(0, Qt.Unchecked)
            construedifi05 = QTreeWidgetItem(construcciones05)
            construedifi05.setFlags(construedifi05.flags() |
Qt.ItemIsUserCheckable)
            construedifi05.setText(0, "Edificaciones CV05")
            construedifi05.setCheckState(0, Qt.Unchecked)
            construcons05 = QTreeWidgetItem(construcciones05)
```

```
            construcons05.setFlags(construcons05.flags() |
Qt.ItemIsUserCheckable)
            construcons05.setText(0, "Construcciones CV05")
            construcons05.setCheckState(0, Qt.Unchecked)
            usos05 = QTreeWidgetItem(basecartografica05)
            usos05.setFlags(usos05.flags() | Qt.ItemIsTristate |
Qt.ItemIsUserCheckable)
            usos05.setText(0, "Usos del suelo")
            usos05.setCheckState(0, Qt.Unchecked)
            usoshidro05 = QTreeWidgetItem(usos05)
            usoshidro05.setFlags(usoshidro05.flags() |
Qt.ItemIsUserCheckable)
            usoshidro05.setText(0, "Hidrografía CV05")
            usoshidro05.setCheckState(0, Qt.Unchecked)
            usosarboles05 = QTreeWidgetItem(usos05)
            usosarboles05.setFlags(usosarboles05.flags() |
Qt.ItemIsUserCheckable)
            usosarboles05.setText(0, "Zonas arboladas CV05")
            usosarboles05.setCheckState(0, Qt.Unchecked)
            usosinfra05 = QTreeWidgetItem(usos05)
            usosinfra05.setFlags(usosinfra05.flags() |
Qt.ItemIsUserCheckable)
            usosinfra05.setText(0, "Infraestructuras viarias CV05")
            usosinfra05.setCheckState(0, Qt.Unchecked)
            ususserv05 = QTreeWidgetItem(usos05)
            ususserv05.setFlags(ususserv05.flags() |
Qt.ItemIsUserCheckable)
            ususserv05.setText(0, "Servicios e instalaciones CV05")
            ususserv05.setCheckState(0, Qt.Unchecked)
            usoscultiv05 = QTreeWidgetItem(usos05)
            usoscultiv05.setFlags(usoscultiv05.flags() |
Qt.ItemIsUserCheckable)
            usoscultiv05.setText(0, "Cultivos CV05")
            usoscultiv05.setCheckState(0, Qt.Unchecked)
            usosentorn05 = QTreeWidgetItem(usos05)
            usosentorn05.setFlags(usosentorn05.flags() |
Qt.ItemIsUserCheckable)
            usosentorn05.setText(0, "Entornos urbanos CV05")
            usosentorn05.setCheckState(0, Qt.Unchecked)
            vias05 = QTreeWidgetItem(basecartografica05)
            vias05.setFlags(vias05.flags() | Qt.ItemIsTristate |
Qt.ItemIsUserCheckable)
            vias05.setText(0, "Vias de Comunicación")
            vias05.setCheckState(0, Qt.Unchecked)
            viasnomen05 = QTreeWidgetItem(vias05)
            viasnomen05.setFlags(viasnomen05.flags() |
Qt.ItemIsUserCheckable)
            viasnomen05.setText(0, "Nomenclatura infraestructura
viaria CV05")
            viasnomen05.setCheckState(0, Qt.Unchecked)
            viasferro05 = QTreeWidgetItem(vias05)
            viasferro05.setFlags(viasferro05.flags() |
Qt.ItemIsUserCheckable)
            viasferro05.setText(0, "Red de ferrocarriles CV05")
            viasferro05.setCheckState(0, Qt.Unchecked)
            viascomu05 = QTreeWidgetItem(vias05)
            viascomu05.setFlags(viascomu05.flags() |
Qt.ItemIsUserCheckable)
            viascomu05.setText(0, "Red de comunicaciones CV05")
            viascomu05.setCheckState(0, Qt.Unchecked)
            basecartografica50 = QTreeWidgetItem(carto)
```

```
            basecartografica50.setText(0, "Base Cartográfica CV50")
            orografia50 = QTreeWidgetItem(basecartografica50)
            orografia50.setFlags(orografia50.flags() |
Qt.ItemIsTristate | Qt.ItemIsUserCheckable)
            orografia50.setText(0, "Orografía  ")
            orografia50.setCheckState(0, Qt.Unchecked)
            orografiacosta50 = QTreeWidgetItem(orografia50)
            orografiacosta50.setFlags(orografiacosta50.flags() |
Qt.ItemIsUserCheckable)
            orografiacosta50.setText(0, "Línea de costa CV50")
            orografiacosta50.setCheckState(0, Qt.Unchecked)
            orografiacurva50 = QTreeWidgetItem(orografia50)
            orografiacurva50.setFlags(orografiacurva50.flags() |
Qt.ItemIsUserCheckable)
            orografiacurva50.setText(0, "Curvas de nivel CV50")
            orografiacurva50.setCheckState(0, Qt.Unchecked)
            hidrografia50 = QTreeWidgetItem(basecartografica50)
            hidrografia50.setFlags(hidrografia50.flags() |
Qt.ItemIsTristate | Qt.ItemIsUserCheckable)
            hidrografia50.setText(0, "Hidrografía")
            hidrografia50.setCheckState(0, Qt.Unchecked)
            hidrocursoa50 = QTreeWidgetItem(hidrografia50)
            hidrocursoa50.setFlags(hidrocursoa50.flags() |
Qt.ItemIsUserCheckable)
            hidrocursoa50.setText(0, "Curso natural CV50")
            hidrocursoa50.setCheckState(0, Qt.Unchecked)
            hidrocurson50 = QTreeWidgetItem(hidrografia50)
            hidrocurson50.setFlags(hidrocurson50.flags() |
Qt.ItemIsUserCheckable)
            hidrocurson50.setText(0, "Curso artificial CV50")
            hidrocurson50.setCheckState(0, Qt.Unchecked)
            hidrolecho50 = QTreeWidgetItem(hidrografia50)
            hidrolecho50.setFlags(hidrolecho50.flags() |
Qt.ItemIsUserCheckable)
            hidrolecho50.setText(0, "Lecho natural CV50")
            hidrolecho50.setCheckState(0, Qt.Unchecked)
            hidroembal50 = QTreeWidgetItem(hidrografia50)
            hidroembal50.setFlags(hidroembal50.flags() |
Qt.ItemIsUserCheckable)
            hidroembal50.setText(0, "Embalse CV50")
            hidroembal50.setCheckState(0, Qt.Unchecked)
            hidrolago50 = QTreeWidgetItem(hidrografia50)
            hidrolago50.setFlags(hidrolago50.flags() |
Qt.ItemIsUserCheckable)
            hidrolago50.setText(0, "Lago CV50")
            hidrolago50.setCheckState(0, Qt.Unchecked)
            hidroisla50 = QTreeWidgetItem(hidrografia50)
            hidroisla50.setFlags(hidroisla50.flags() |
Qt.ItemIsUserCheckable)
            hidroisla50.setText(0, "Isla CV50")
            hidroisla50.setCheckState(0, Qt.Unchecked)
            edificacion50 = QTreeWidgetItem(basecartografica50)
            edificacion50.setFlags(edificacion50.flags() |
Qt.ItemIsTristate | Qt.ItemIsUserCheckable)
            edificacion50.setText(0, "Edificaciones y construcciones")
            edificacion50.setCheckState(0, Qt.Unchecked)
            edifiedi50 = QTreeWidgetItem(edificacion50)
            edifiedi50.setFlags(edifiedi50.flags() |
Qt.ItemIsUserCheckable)
            edifiedi50.setText(0, "Edificaciones CV50")
            edifiedi50.setCheckState(0, Qt.Unchecked)
```

```
            edifiedip50 = QTreeWidgetItem(edificacion50)
            edifiedip50.setFlags(edifiedip50.flags() |
Qt.ItemIsUserCheckable)
            edifiedip50.setText(0, "Edificaciones puntuales CV50")
            edifiedip50.setCheckState(0, Qt.Unchecked)
            edifipisci50 = QTreeWidgetItem(edificacion50)
            edifipisci50.setFlags(edifipisci50.flags() |
Qt.ItemIsUserCheckable)
            edifipisci50.setText(0, "Piscina / Balsa CV50")
            edifipisci50.setCheckState(0, Qt.Unchecked)
            edifipiscip50 = QTreeWidgetItem(edificacion50)
            edifipiscip50.setFlags(edifipiscip50.flags() |
Qt.ItemIsUserCheckable)
            edifipiscip50.setText(0, "Piscina / Balsa puntual CV50")
            edifipiscip50.setCheckState(0, Qt.Unchecked)
            edifiruina50 = QTreeWidgetItem(edificacion50)
            edifiruina50.setFlags(edifiruina50.flags() |
Qt.ItemIsUserCheckable)
            edifiruina50.setText(0, "Edificación en ruinas CV50")
            edifiruina50.setCheckState(0, Qt.Unchecked)
            edifiruinap50 = QTreeWidgetItem(edificacion50)
            edifiruinap50.setFlags(edifiruinap50.flags() |
Qt.ItemIsUserCheckable)
            edifiruinap50.setText(0, "Edificación en ruinas puntual
CV50")
            edifiruinap50.setCheckState(0, Qt.Unchecked)
            edifextrac50 = QTreeWidgetItem(edificacion50)
            edifextrac50.setFlags(edifextrac50.flags() |
Qt.ItemIsUserCheckable)
            edifextrac50.setText(0, "Actividad extractiva CV50")
            edifextrac50.setCheckState(0, Qt.Unchecked)
            edifextracp50 = QTreeWidgetItem(edificacion50)
            edifextracp50.setFlags(edifextracp50.flags() |
Qt.ItemIsUserCheckable)
            edifextracp50.setText(0, "Actividad extractiva puntual
CV50")
            edifextracp50.setCheckState(0, Qt.Unchecked)
            edifdeposit50 = QTreeWidgetItem(edificacion50)
            edifdeposit50.setFlags(edifdeposit50.flags() |
Qt.ItemIsUserCheckable)
            edifdeposit50.setText(0, "Depósito")
            edifdeposit50.setCheckState(0, Qt.Unchecked)
            edifobrac50 = QTreeWidgetItem(edificacion50)
            edifobrac50.setFlags(edifobrac50.flags() |
Qt.ItemIsUserCheckable)
            edifobrac50.setText(0, "Obra de contención")
            edifobrac50.setCheckState(0, Qt.Unchecked)
            edifinverna50 = QTreeWidgetItem(edificacion50)
            edifinverna50.setFlags(edifinverna50.flags() |
Qt.ItemIsUserCheckable)
            edifinverna50.setText(0, "Invernadero")
            edifinverna50.setCheckState(0, Qt.Unchecked)
            edifpatio50 = QTreeWidgetItem(edificacion50)
            edifpatio50.setFlags(edifpatio50.flags() |
Qt.ItemIsUserCheckable)
            edifpatio50.setText(0, "Patio")
            edifpatio50.setCheckState(0, Qt.Unchecked)
            edifmuralla50 = QTreeWidgetItem(edificacion50)
            edifmuralla50.setFlags(edifmuralla50.flags() |
Qt.ItemIsUserCheckable)
            edifmuralla50.setText(0, "Muralla histórica")
```

```
            edifmuralla50.setCheckState(0, Qt.Unchecked)
            instalacion50 = QTreeWidgetItem(basecartografica50)
            instalacion50.setFlags(instalacion50.flags() |
Qt.ItemIsTristate | Qt.ItemIsUserCheckable)
            instalacion50.setText(0, "Instalaciones y equipamientos")
            instalacion50.setCheckState(0, Qt.Unchecked)
            instadota50 = QTreeWidgetItem(instalacion50)
            instadota50.setFlags(instadota50.flags() |
Qt.ItemIsUserCheckable)
            instadota50.setText(0, "Espacio dotacional")
            instadota50.setCheckState(0, Qt.Unchecked)
            instacircu50 = QTreeWidgetItem(instalacion50)
            instacircu50.setFlags(instacircu50.flags() |
Qt.ItemIsUserCheckable)
            instacircu50.setText(0, "Circuito")
            instacircu50.setCheckState(0, Qt.Unchecked)
            instapoli50 = QTreeWidgetItem(instalacion50)
            instapoli50.setFlags(instapoli50.flags() |
Qt.ItemIsUserCheckable)
            instapoli50.setText(0, "Instalación de transporte
poligonal")
            instapoli50.setCheckState(0, Qt.Unchecked)
            instalinea50 = QTreeWidgetItem(instalacion50)
            instalinea50.setFlags(instalinea50.flags() |
Qt.ItemIsUserCheckable)
            instalinea50.setText(0, "Instalación de transporte
lineal")
            instalinea50.setCheckState(0, Qt.Unchecked)
            instapunt50 = QTreeWidgetItem(instalacion50)
            instapunt50.setFlags(instapunt50.flags() |
Qt.ItemIsUserCheckable)
            instapunt50.setText(0, "Instalación de transporte
puntual")
            instapunt50.setCheckState(0, Qt.Unchecked)
            instadotpun50 = QTreeWidgetItem(instalacion50)
            instadotpun50.setFlags(instadotpun50.flags() |
Qt.ItemIsUserCheckable)
            instadotpun50.setText(0, "Espacio dotacional puntual")
            instadotpun50.setCheckState(0, Qt.Unchecked)
            instaelect50 = QTreeWidgetItem(instalacion50)
            instaelect50.setFlags(instaelect50.flags() |
Qt.ItemIsUserCheckable)
            instaelect50.setText(0, "Línea eléctrica")
            instaelect50.setCheckState(0, Qt.Unchecked)
            instavertic50 = QTreeWidgetItem(instalacion50)
            instavertic50.setFlags(instavertic50.flags() |
Qt.ItemIsUserCheckable)
            instavertic50.setText(0, "Vértices geodésicos")
            instavertic50.setCheckState(0, Qt.Unchecked)
            viascomu50 = QTreeWidgetItem(basecartografica50)
            viascomu50.setFlags(viascomu50.flags() | Qt.ItemIsTristate
| Qt.ItemIsUserCheckable)
            viascomu50.setText(0, "Vias de Comunicación")
            viascomu50.setCheckState(0, Qt.Unchecked)
            instared50 = QTreeWidgetItem(viascomu50)
            instared50.setFlags(instared50.flags() |
Qt.ItemIsUserCheckable)
            instared50.setText(0, "Red de Comunicaciones")
            instared50.setCheckState(0, Qt.Unchecked)
            instaferroc50 = QTreeWidgetItem(viascomu50)
            instaferroc50.setFlags(instaferroc50.flags() |
```

```
Qt.ItemIsUserCheckable)
            instaferroc50.setText(0, "Ferrocarril")
            instaferroc50.setCheckState(0, Qt.Unchecked)
            instaportal0 = QTreeWidgetItem(viascomu50)
            instaportal0.setFlags(instaportal0.flags() |
Qt.ItemIsUserCheckable)
            instaportal0.setText(0, "Portales/PKs")
            instaportal0.setCheckState(0, Qt.Unchecked)
            instabocat0 = QTreeWidgetItem(viascomu50)
            instabocat0.setFlags(instabocat0.flags() |
Qt.ItemIsUserCheckable)
            instabocat0.setText(0, "Boca de túnel")
            instabocat0.setCheckState(0, Qt.Unchecked)
            instaredele50 = QTreeWidgetItem(viascomu50)
            instaredele50.setFlags(instaredele50.flags() |
Qt.ItemIsUserCheckable)
            instaredele50.setText(0, "Red viaria elevada")
            instaredele50.setCheckState(0, Qt.Unchecked)
            instaferrel50 = QTreeWidgetItem(viascomu50)
            instaferrel50.setFlags(instaferrel50.flags() |
Qt.ItemIsUserCheckable)
            instaferrel50.setText(0, "Ferrocarril elevado")
            instaferrel50.setCheckState(0, Qt.Unchecked)
            instaviarisub50 = QTreeWidgetItem(viascomu50)
            instaviarisub50.setFlags(instaviarisub50.flags() |
Qt.ItemIsUserCheckable)
            instaviarisub50.setText(0, "Red viaria subterránea")
            instaviarisub50.setCheckState(0, Qt.Unchecked)
            instaviarisub50 = QTreeWidgetItem(viascomu50)
            instaviarisub50.setFlags(instaviarisub50.flags() |
Qt.ItemIsUserCheckable)
            instaviarisub50.setText(0, "Ferrocarril subterráneo")
            instaviarisub50.setCheckState(0, Qt.Unchecked)
            basecartografica100 = QTreeWidgetItem(carto)
            basecartografica100.setText(0, "Base Cartogràfica CV100")
            orografia100 = QTreeWidgetItem(basecartografica100)
            orografia100.setFlags(orografia100.flags() |
Qt.ItemIsTristate | Qt.ItemIsUserCheckable)
            orografia100.setText(0, "Orografía")
            orografia100.setCheckState(0, Qt.Unchecked)
            costa100 = QTreeWidgetItem(orografia100)
            costa100.setFlags(costa100.flags() |
Qt.ItemIsUserCheckable)
            costa100.setText(0, "Línea de costa CV100")
            costa100.setCheckState(0, Qt.Unchecked)
            curvas100 = QTreeWidgetItem(orografia100)
            curvas100.setFlags(curvas100.flags() |
Qt.ItemIsUserCheckable)
            curvas100.setText(0, "Curvas de nivel CV100")
            curvas100.setCheckState(0, Qt.Unchecked)
            cota100 = QTreeWidgetItem(orografia100)
            cota100.setFlags(cota100.flags() | Qt.ItemIsUserCheckable)
            cota100.setText(0, "Puntos de cota CV100")
            cota100.setCheckState(0, Qt.Unchecked)
            hidrografia100 = QTreeWidgetItem(basecartografica100)
            hidrografia100.setFlags(hidrografia100.flags() |
Qt.ItemIsTristate | Qt.ItemIsUserCheckable)
            hidrografia100.setText(0, "Hidrografia")
            hidrografia100.setCheckState(0, Qt.Unchecked)
            curson100 = QTreeWidgetItem(hidrografia100)
            curson100.setFlags(curson100.flags() |
```

```
Qt.ItemIsUserCheckable)
            curson100.setText(0, "Curso natural CV100")
            curson100.setCheckState(0, Qt.Unchecked)
            lecho100 = QTreeWidgetItem(hidrografia100)
            lecho100.setFlags(lecho100.flags() |
Qt.ItemIsUserCheckable)
            lecho100.setText(0, "Lecho natural CV100")
            lecho100.setCheckState(0, Qt.Unchecked)
            lago100 = QTreeWidgetItem(hidrografia100)
            lago100.setFlags(lago100.flags() | Qt.ItemIsUserCheckable)
            lago100.setText(0, "Lago CV100")
            lago100.setCheckState(0, Qt.Unchecked)
            embalse100 = QTreeWidgetItem(hidrografia100)
            embalse100.setFlags(embalse100.flags() |
Qt.ItemIsUserCheckable)
            embalse100.setText(0, "Embalse CV100")
            embalse100.setCheckState(0, Qt.Unchecked)
            isla100 = QTreeWidgetItem(hidrografia100)
            isla100.setFlags(isla100.flags() | Qt.ItemIsUserCheckable)
            isla100.setText(0, "Isla CV100")
            isla100.setCheckState(0, Qt.Unchecked)
            edifica100 = QTreeWidgetItem(basecartografica100)
            edifica100.setFlags(edifica100.flags() | Qt.ItemIsTristate
| Qt.ItemIsUserCheckable)
            edifica100.setText(0, "Edificaciones y construcciones")
            edifica100.setCheckState(0, Qt.Unchecked)
            edificaed100 = QTreeWidgetItem(edifica100)
            edificaed100.setFlags(edificaed100.flags() |
Qt.ItemIsUserCheckable)
            edificaed100.setText(0, "Edificación CV100")
            edificaed100.setCheckState(0, Qt.Unchecked)
            edificaedp100 = QTreeWidgetItem(edifica100)
            edificaedp100.setFlags(edificaedp100.flags() |
Qt.ItemIsUserCheckable)
            edificaedp100.setText(0, "Edificación puntual CV100")
            edificaedp100.setCheckState(0, Qt.Unchecked)
            piscina100 = QTreeWidgetItem(edifica100)
            piscina100.setFlags(piscina100.flags() |
Qt.ItemIsUserCheckable)
            piscina100.setText(0, "Piscina/Balsa CV100")
            piscina100.setCheckState(0, Qt.Unchecked)
            piscinap100 = QTreeWidgetItem(edifica100)
            piscinap100.setFlags(piscinap100.flags() |
Qt.ItemIsUserCheckable)
            piscinap100.setText(0, "Piscina/Balsa puntual CV100")
            piscinap100.setCheckState(0, Qt.Unchecked)
            edificaedr100 = QTreeWidgetItem(edifica100)
            edificaedr100.setFlags(edificaedr100.flags() |
Qt.ItemIsUserCheckable)
            edificaedr100.setText(0, "Edificación en ruinas CV100")
            edificaedr100.setCheckState(0, Qt.Unchecked)
            edificaedrp100 = QTreeWidgetItem(edifica100)
            edificaedrp100.setFlags(edificaedrp100.flags() |
Qt.ItemIsUserCheckable)
            edificaedrp100.setText(0, "Edificación en ruinas puntual
CV100")
            edificaedrp100.setCheckState(0, Qt.Unchecked)
            actext100 = QTreeWidgetItem(edifica100)
            actext100.setFlags(actext100.flags() |
Qt.ItemIsUserCheckable)
            actext100.setText(0, "Actividad extractiva CV100")
```

```
            actext100.setCheckState(0, Qt.Unchecked)
            actextp100 = QTreeWidgetItem(edifica100)
            actextp100.setFlags(actextp100.flags() |
Qt.ItemIsUserCheckable)
            actextp100.setText(0, "Actividad extractiva puntual
CV100")
            actextp100.setCheckState(0, Qt.Unchecked)
            obracon100 = QTreeWidgetItem(edifica100)
            obracon100.setFlags(obracon100.flags() |
Qt.ItemIsUserCheckable)
            obracon100.setText(0, "Obra de contención CV100")
            obracon100.setCheckState(0, Qt.Unchecked)
            inverna100 = QTreeWidgetItem(edifica100)
            inverna100.setFlags(inverna100.flags() |
Qt.ItemIsUserCheckable)
            inverna100.setText(0, "Invernadero CV100")
            inverna100.setCheckState(0, Qt.Unchecked)
            patio100 = QTreeWidgetItem(edifica100)
            patio100.setFlags(patio100.flags() |
Qt.ItemIsUserCheckable)
            patio100.setText(0, "Patio CV100")
            patio100.setCheckState(0, Qt.Unchecked)
            muralla100 = QTreeWidgetItem(edifica100)
            muralla100.setFlags(muralla100.flags() |
Qt.ItemIsUserCheckable)
            muralla100.setText(0, "Muralla histórica CV100")
            muralla100.setCheckState(0, Qt.Unchecked)
            instala100 = QTreeWidgetItem(basecartografica100)
            instala100.setFlags(instala100.flags() | Qt.ItemIsTristate
| Qt.ItemIsUserCheckable)
            instala100.setText(0, "Instalaciones y equipamientos")
            instala100.setCheckState(0, Qt.Unchecked)
            instalinea100 = QTreeWidgetItem(instala100)
            instalinea100.setFlags(instalinea100.flags() |
Qt.ItemIsUserCheckable)
            instalinea100.setText(0, "Línea eléctrica CV100")
            instalinea100.setCheckState(0, Qt.Unchecked)
            vertices100 = QTreeWidgetItem(instala100)
            vertices100.setFlags(vertices100.flags() |
Qt.ItemIsUserCheckable)
            vertices100.setText(0, "Vértices geodésicos CV100")
            vertices100.setCheckState(0, Qt.Unchecked)
            dotapunt100 = QTreeWidgetItem(instala100)
            dotapunt100.setFlags(dotapunt100.flags() |
Qt.ItemIsUserCheckable)
            dotapunt100.setText(0, "Espacio dotacional puntual CV100")
            dotapunt100.setCheckState(0, Qt.Unchecked)
            transpunt100 = QTreeWidgetItem(instala100)
            transpunt100.setFlags(transpunt100.flags() |
Qt.ItemIsUserCheckable)
            transpunt100.setText(0, "Instalación de transporte puntual
CV100")
            transpunt100.setCheckState(0, Qt.Unchecked)
            deposito100 = QTreeWidgetItem(instala100)
            deposito100.setFlags(deposito100.flags() |
Qt.ItemIsUserCheckable)
            deposito100.setText(0, "Depósito CV100")
            deposito100.setCheckState(0, Qt.Unchecked)
            translin100 = QTreeWidgetItem(instala100)
            translin100.setFlags(translin100.flags() |
Qt.ItemIsUserCheckable)
```

```
            translin100.setText(0, "Instalación de transporte lineal
CV100")
            translin100.setCheckState(0, Qt.Unchecked)
            transpol100 = QTreeWidgetItem(instala100)
            transpol100.setFlags(transpol100.flags() |
Qt.ItemIsUserCheckable)
            transpol100.setText(0, "Instalación de transporte
poligonal CV100")
            transpol100.setCheckState(0, Qt.Unchecked)
            espadot100 = QTreeWidgetItem(instala100)
            espadot100.setFlags(espadot100.flags() |
Qt.ItemIsUserCheckable)
            espadot100.setText(0, "Espacio dotacional CV100")
            espadot100.setCheckState(0, Qt.Unchecked)
            comunicacion100 = QTreeWidgetItem(basecartografica100)
            comunicacion100.setFlags(comunicacion100.flags() |
Qt.ItemIsTristate | Qt.ItemIsUserCheckable)
            comunicacion100.setText(0, "Vías de comunicación")
            comunicacion100.setCheckState(0, Qt.Unchecked)
            comured100 = QTreeWidgetItem(comunicacion100)
            comured100.setFlags(comured100.flags() |
Qt.ItemIsUserCheckable)
            comured100.setText(0, "Red viaria CV100")
            comured100.setCheckState(0, Qt.Unchecked)
            ferro100 = QTreeWidgetItem(comunicacion100)
            ferro100.setFlags(ferro100.flags() |
Qt.ItemIsUserCheckable)
            ferro100.setText(0, "Ferrocarril CV100")
            ferro100.setCheckState(0, Qt.Unchecked)
            tunel100 = QTreeWidgetItem(comunicacion100)
            tunel100.setFlags(tunel100.flags() |
Qt.ItemIsUserCheckable)
            tunel100.setText(0, "Boca de túnel CV100")
            tunel100.setCheckState(0, Qt.Unchecked)
            comurede100 = QTreeWidgetItem(comunicacion100)
            comurede100.setFlags(comurede100.flags() |
Qt.ItemIsUserCheckable)
            comurede100.setText(0, "Red viaria elevada CV100")
            comurede100.setCheckState(0, Qt.Unchecked)
            ferroe100 = QTreeWidgetItem(comunicacion100)
            ferroe100.setFlags(ferroe100.flags() |
Qt.ItemIsUserCheckable)
            ferroe100.setText(0, "Ferrocarril elevado CV100")
            ferroe100.setCheckState(0, Qt.Unchecked)
            comureds100 = QTreeWidgetItem(comunicacion100)
            comureds100.setFlags(comureds100.flags() |
Qt.ItemIsUserCheckable)
            comureds100.setText(0, "Red viaria subterránea CV100")
            comureds100.setCheckState(0, Qt.Unchecked)
            ferros100 = QTreeWidgetItem(comunicacion100)
            ferros100.setFlags(ferros100.flags() |
Qt.ItemIsUserCheckable)
            ferros100.setText(0, "Ferrocarril subterráneo CV100")
            ferros100.setCheckState(0, Qt.Unchecked)
            fechasbases = QTreeWidgetItem(carto)
            fechasbases.setText(0, "Fechas Bases cartográficas")
            fechasvuelos = QTreeWidgetItem(fechasbases)
            fechasvuelos.setFlags(fechasvuelos.flags() |
Qt.ItemIsUserCheckable)
            fechasvuelos.setText(0, "Fechas de vuelo")
            fechasvuelos.setCheckState(0, Qt.Unchecked)
```

```
            redtransp = QTreeWidgetItem(carto)
            redtransp.setText(0, "Red de Transportes")
            redcarretera = QTreeWidgetItem(redtransp)
            redcarretera.setFlags(redcarretera.flags() |
Qt.ItemIsTristate | Qt.ItemIsUserCheckable)
            redcarretera.setText(0, "Transporte Carretera")
            redcarretera.setCheckState(0, Qt.Unchecked)
            sistviari = QTreeWidgetItem(redcarretera)
            sistviari.setFlags(sistviari.flags() | Qt.ItemIsTristate |
Qt.ItemIsUserCheckable)
            sistviari.setText(0, "Sistema viario")
            sistviari.setCheckState(0, Qt.Unchecked)
            cataviario = QTreeWidgetItem(sistviari)
            cataviario.setFlags(cataviario.flags() | Qt.ItemIsTristate
| Qt.ItemIsUserCheckable)
            cataviario.setText(0, "Catálogo Viario")
            cataviario.setCheckState(0, Qt.Unchecked)
            cataviario1 = QTreeWidgetItem(cataviario)
            cataviario1.setFlags(cataviario1.flags() |
Qt.ItemIsUserCheckable)
            cataviario1.setText(0, "Catálogo viario escala 1")
            cataviario1.setCheckState(0, Qt.Unchecked)
            cataviario2 = QTreeWidgetItem(cataviario)
            cataviario2.setFlags(cataviario1.flags() |
Qt.ItemIsUserCheckable)
            cataviario2.setText(0, "Catálogo viario escala 2")
            cataviario2.setCheckState(0, Qt.Unchecked)
            cataviario3 = QTreeWidgetItem(cataviario)
            cataviario3.setFlags(cataviario3.flags() |
Qt.ItemIsUserCheckable)
            cataviario3.setText(0, "Catálogo viario escala 3")
            cataviario3.setCheckState(0, Qt.Unchecked)
            cataviario4 = QTreeWidgetItem(cataviario)
            cataviario4.setFlags(cataviario4.flags() |
Qt.ItemIsUserCheckable)
            cataviario4.setText(0, "Catálogo viario escala 4")
            cataviario4.setCheckState(0, Qt.Unchecked)
            puntokil = QTreeWidgetItem(sistviari)
            puntokil.setFlags(puntokil.flags() |
Qt.ItemIsUserCheckable)
            puntokil.setText(0, "Puntos kilométricos")
            puntokil.setCheckState(0, Qt.Unchecked)
            caminos = QTreeWidgetItem(sistviari)
            caminos.setFlags(puntokil.flags() |
Qt.ItemIsUserCheckable)
            caminos.setText(0, "Caminos")
            caminos.setCheckState(0, Qt.Unchecked)
            urbana = QTreeWidgetItem(sistviari)
            urbana.setFlags(urbana.flags() | Qt.ItemIsTristate |
Qt.ItemIsUserCheckable)
            urbana.setText(0, "Urbana")
            urbana.setCheckState(0, Qt.Unchecked)
            listacalles = QTreeWidgetItem(urbana)
            listacalles.setFlags(listacalles.flags() |
Qt.ItemIsUserCheckable)
            listacalles.setText(0, "Listado de calles")
            listacalles.setCheckState(0, Qt.Unchecked)
            portal = QTreeWidgetItem(urbana)
            portal.setFlags(portal.flags() | Qt.ItemIsUserCheckable)
            portal.setText(0, "Portal")
            portal.setCheckState(0, Qt.Unchecked)
```

```
            infraestruc = QTreeWidgetItem(redcarretera)
            infraestruc.setFlags(infraestruc.flags() |
Qt.ItemIsTristate | Qt.ItemIsUserCheckable)
            infraestruc.setText(0, "Infraestructuras e instalaciones")
            infraestruc.setCheckState(0, Qt.Unchecked)
            estautob = QTreeWidgetItem(infraestruc)
            estautob.setFlags(estautob.flags() |
Qt.ItemIsUserCheckable)
            estautob.setText(0, "Estación de autobuses")
            estautob.setCheckState(0, Qt.Unchecked)
            peaje = QTreeWidgetItem(infraestruc)
            peaje.setFlags(peaje.flags() | Qt.ItemIsUserCheckable)
            peaje.setText(0, "Peaje")
            peaje.setCheckState(0, Qt.Unchecked)
            areaserv = QTreeWidgetItem(infraestruc)
            areaserv.setFlags(areaserv.flags() |
Qt.ItemIsUserCheckable)
            areaserv.setText(0, "Áreas de servicio")
            areaserv.setCheckState(0, Qt.Unchecked)
            areaservin = QTreeWidgetItem(infraestruc)
            areaservin.setFlags(areaservin.flags() |
Qt.ItemIsUserCheckable)
            areaservin.setText(0, "Aparcamiento de servicios viarios
invernal")
            areaservin.setCheckState(0, Qt.Unchecked)
            puente = QTreeWidgetItem(infraestruc)
            puente.setFlags(puente.flags() | Qt.ItemIsUserCheckable)
            puente.setText(0, "Puente")
            puente.setCheckState(0, Qt.Unchecked)




        if self.rdb_val.isChecked() == True:  # Create the fahther and
child nodes in valencian
            carto.setText(0, "Cartografia de referència")
            # carto.setFlags(carto.flags() | Qt.ItemIsTristate |
Qt.ItemIsUserCheckable)
            toponimia = QTreeWidgetItem(carto)
            toponimia.setFlags(toponimia.flags() |
Qt.ItemIsUserCheckable)
            toponimia.setText(0, "Nomenclàtor Toponímic Valencià")
            toponimia.setCheckState(0, Qt.Unchecked)
            basecartografica05 = QTreeWidgetItem(carto)
            basecartografica05.setText(0, "Base Cartogràfica CV05")
            orografia05 = QTreeWidgetItem(basecartografica05)
            orografia05.setFlags(orografia05.flags() |
Qt.ItemIsTristate | Qt.ItemIsUserCheckable)
            orografia05.setText(0, "Orografia")
            orografia05.setCheckState(0, Qt.Unchecked)
            puntoscota = QTreeWidgetItem(orografia05)
            puntoscota.setFlags(puntoscota.flags() |
Qt.ItemIsUserCheckable)
            puntoscota.setText(0, "Punts de cota CV05")
            puntoscota.setCheckState(0, Qt.Unchecked)
            orolineas = QTreeWidgetItem(orografia05)
            orolineas.setFlags(orolineas.flags() |
Qt.ItemIsUserCheckable)
            orolineas.setText(0, "Orografia línies CV05")
            orolineas.setCheckState(0, Qt.Unchecked)
```

```
            curvasnivel = QTreeWidgetItem(orografia05)
            curvasnivel.setFlags(orolineas.flags() |
Qt.ItemIsUserCheckable)
            curvasnivel.setText(0, "Corbes de nivell CV05")
            curvasnivel.setCheckState(0, Qt.Unchecked)
            hidrografia05 = QTreeWidgetItem(basecartografica05)
            hidrografia05.setFlags(hidrografia05.flags() |
Qt.ItemIsTristate | Qt.ItemIsUserCheckable)
            hidrografia05.setText(0, "Hidrografia")
            hidrografia05.setCheckState(0, Qt.Unchecked)
            hidropuntos05 = QTreeWidgetItem(hidrografia05)
            hidropuntos05.setFlags(orolineas.flags() |
Qt.ItemIsUserCheckable)
            hidropuntos05.setText(0, "Hidrografia puntual CV05")
            hidropuntos05.setCheckState(0, Qt.Unchecked)
            hidrolineas05 = QTreeWidgetItem(hidrografia05)
            hidrolineas05.setFlags(orolineas.flags() |
Qt.ItemIsUserCheckable)
            hidrolineas05.setText(0, "Hidrografia lineal CV05")
            hidrolineas05.setCheckState(0, Qt.Unchecked)
            construcciones05 = QTreeWidgetItem(basecartografica05)
            construcciones05.setFlags(construcciones05.flags() |
Qt.ItemIsTristate | Qt.ItemIsUserCheckable)
            construcciones05.setText(0, "Edificacions i
construccions")
            construcciones05.setCheckState(0, Qt.Unchecked)
            construpuntos05 = QTreeWidgetItem(construcciones05)
            construpuntos05.setFlags(construpuntos05.flags() |
Qt.ItemIsUserCheckable)
            construpuntos05.setText(0, "Construccions puntuals CV05")
            construpuntos05.setCheckState(0, Qt.Unchecked)
            construservi05 = QTreeWidgetItem(construcciones05)
            construservi05.setFlags(construservi05.flags() |
Qt.ItemIsUserCheckable)
            construservi05.setText(0, "Serveis i instal·lacions CV05")
            construservi05.setCheckState(0, Qt.Unchecked)
            construlinea05 = QTreeWidgetItem(construcciones05)
            construlinea05.setFlags(construlinea05.flags() |
Qt.ItemIsUserCheckable)
            construlinea05.setText(0, "Construccions lineals CV05")
            construlinea05.setCheckState(0, Qt.Unchecked)
            construedifi05 = QTreeWidgetItem(construcciones05)
            construedifi05.setFlags(construedifi05.flags() |
Qt.ItemIsUserCheckable)
            construedifi05.setText(0, "Edificacions CV05")
            construedifi05.setCheckState(0, Qt.Unchecked)
            construcons05 = QTreeWidgetItem(construcciones05)
            construcons05.setFlags(construcons05.flags() |
Qt.ItemIsUserCheckable)
            construcons05.setText(0, "Construccions CV05")
            construcons05.setCheckState(0, Qt.Unchecked)
            usos05 = QTreeWidgetItem(basecartografica05)
            usos05.setFlags(usos05.flags() | Qt.ItemIsTristate |
Qt.ItemIsUserCheckable)
            usos05.setText(0, "Usos del sòl")
            usos05.setCheckState(0, Qt.Unchecked)
            usoshidro05 = QTreeWidgetItem(usos05)
            usoshidro05.setFlags(usoshidro05.flags() |
Qt.ItemIsUserCheckable)
            usoshidro05.setText(0, "Hidrografia CV05")
            usoshidro05.setCheckState(0, Qt.Unchecked)
```

```
            usosarboles05 = QTreeWidgetItem(usos05)
            usosarboles05.setFlags(usosarboles05.flags() |
Qt.ItemIsUserCheckable)
            usosarboles05.setText(0, "Zones amb arbres CV05")
            usosarboles05.setCheckState(0, Qt.Unchecked)
            ususinfra05 = QTreeWidgetItem(usos05)
            ususinfra05.setFlags(ususinfra05.flags() |
Qt.ItemIsUserCheckable)
            ususinfra05.setText(0, "Infraestructures viàries CV05")
            ususinfra05.setCheckState(0, Qt.Unchecked)
            usosserv05 = QTreeWidgetItem(usos05)
            usosserv05.setFlags(usosserv05.flags() |
Qt.ItemIsUserCheckable)
            usosserv05.setText(0, "Serveis i instal·lacions CV05")
            usosserv05.setCheckState(0, Qt.Unchecked)
            usoscultiv05 = QTreeWidgetItem(usos05)
            usoscultiv05.setFlags(usoscultiv05.flags() |
Qt.ItemIsUserCheckable)
            usoscultiv05.setText(0, "Cultius CV05")
            usoscultiv05.setCheckState(0, Qt.Unchecked)
            usosentorn05 = QTreeWidgetItem(usos05)
            usosentorn05.setFlags(usosentorn05.flags() |
Qt.ItemIsUserCheckable)
            usosentorn05.setText(0, "Entorns urbans CV05")
            usosentorn05.setCheckState(0, Qt.Unchecked)
            vias05 = QTreeWidgetItem(basecartografica05)
            vias05.setFlags(vias05.flags() | Qt.ItemIsTristate |
Qt.ItemIsUserCheckable)
            vias05.setText(0, "Vies de Comunicació")
            vias05.setCheckState(0, Qt.Unchecked)
            viasnomen05 = QTreeWidgetItem(vias05)
            viasnomen05.setFlags(viasnomen05.flags() |
Qt.ItemIsUserCheckable)
            viasnomen05.setText(0, "Nomenclatura infraestructura
viària CV05")
            viasnomen05.setCheckState(0, Qt.Unchecked)
            viasferro05 = QTreeWidgetItem(vias05)
            viasferro05.setFlags(viasferro05.flags() |
Qt.ItemIsUserCheckable)
            viasferro05.setText(0, "Xarxa de ferrocarrils CV05")
            viasferro05.setCheckState(0, Qt.Unchecked)
            viascomu05 = QTreeWidgetItem(vias05)
            viascomu05.setFlags(viascomu05.flags() |
Qt.ItemIsUserCheckable)
            viascomu05.setText(0, "Xarxa de comunicacions CV05")
            viascomu05.setCheckState(0, Qt.Unchecked)
            basecartografica50 = QTreeWidgetItem(carto)
            basecartografica50.setText(0, "Base Cartogràfica CV50")
            orografia50 = QTreeWidgetItem(basecartografica50)
            orografia50.setFlags(orografia50.flags() |
Qt.ItemIsTristate | Qt.ItemIsUserCheckable)
            orografia50.setText(0, "Orografia  ")
            orografia50.setCheckState(0, Qt.Unchecked)
            orografiacosta50 = QTreeWidgetItem(orografia50)
            orografiacosta50.setFlags(orografiacosta50.flags() |
Qt.ItemIsUserCheckable)
            orografiacosta50.setText(0, "Línia de costa CV50")
            orografiacosta50.setCheckState(0, Qt.Unchecked)
            orografiacurva50 = QTreeWidgetItem(orografia50)
            orografiacurva50.setFlags(orografiacurva50.flags() |
Qt.ItemIsUserCheckable)
```

```
            orografiacurva50.setText(0, "Corbes de nivell CV50")
            orografiacurva50.setCheckState(0, Qt.Unchecked)
            hidrografia50 = QTreeWidgetItem(basecartografica50)
            hidrografia50.setFlags(hidrografia50.flags() |
Qt.ItemIsTristate | Qt.ItemIsUserCheckable)
            hidrografia50.setText(0, "Hidrografia")
            hidrografia50.setCheckState(0, Qt.Unchecked)
            hidrocursoa50 = QTreeWidgetItem(hidrografia50)
            hidrocursoa50.setFlags(hidrocursoa50.flags() |
Qt.ItemIsUserCheckable)
            hidrocursoa50.setText(0, "Curs natural CV50")
            hidrocursoa50.setCheckState(0, Qt.Unchecked)
            hidrocurson50 = QTreeWidgetItem(hidrografia50)
            hidrocurson50.setFlags(hidrocurson50.flags() |
Qt.ItemIsUserCheckable)
            hidrocurson50.setText(0, "Curs artificial CV50")
            hidrocurson50.setCheckState(0, Qt.Unchecked)
            hidrolecho50 = QTreeWidgetItem(hidrografia50)
            hidrolecho50.setFlags(hidrolecho50.flags() |
Qt.ItemIsUserCheckable)
            hidrolecho50.setText(0, "Llit natural CV50")
            hidrolecho50.setCheckState(0, Qt.Unchecked)
            hidroembal50 = QTreeWidgetItem(hidrografia50)
            hidroembal50.setFlags(hidroembal50.flags() |
Qt.ItemIsUserCheckable)
            hidroembal50.setText(0, "Embassament CV50")
            hidroembal50.setCheckState(0, Qt.Unchecked)
            hidrolago50 = QTreeWidgetItem(hidrografia50)
            hidrolago50.setFlags(hidrolago50.flags() |
Qt.ItemIsUserCheckable)
            hidrolago50.setText(0, "Llac CV50")
            hidrolago50.setCheckState(0, Qt.Unchecked)
            hidroisla50 = QTreeWidgetItem(hidrografia50)
            hidroisla50.setFlags(hidroisla50.flags() |
Qt.ItemIsUserCheckable)
            hidroisla50.setText(0, "Illa CV50")
            hidroisla50.setCheckState(0, Qt.Unchecked)
            edificacion50 = QTreeWidgetItem(basecartografica50)
            edificacion50.setFlags(edificacion50.flags() |
Qt.ItemIsTristate | Qt.ItemIsUserCheckable)
            edificacion50.setText(0, "Edificacions i construccions")
            edificacion50.setCheckState(0, Qt.Unchecked)
            edifiedi50 = QTreeWidgetItem(edificacion50)
            edifiedi50.setFlags(edifiedi50.flags() |
Qt.ItemIsUserCheckable)
            edifiedi50.setText(0, "Edificacions CV50")
            edifiedi50.setCheckState(0, Qt.Unchecked)
            edifiedip50 = QTreeWidgetItem(edificacion50)
            edifiedip50.setFlags(edifiedip50.flags() |
Qt.ItemIsUserCheckable)
            edifiedip50.setText(0, "Edificacions puntuals CV50")
            edifiedip50.setCheckState(0, Qt.Unchecked)
            edifipisci50 = QTreeWidgetItem(edificacion50)
            edifipisci50.setFlags(edifipisci50.flags() |
Qt.ItemIsUserCheckable)
            edifipisci50.setText(0, "Piscina / Bassa CV50")
            edifipisci50.setCheckState(0, Qt.Unchecked)
            edifipiscip50 = QTreeWidgetItem(edificacion50)
            edifipiscip50.setFlags(edifipiscip50.flags() |
Qt.ItemIsUserCheckable)
            edifipiscip50.setText(0, "Piscina / Bassa puntual CV50")
```

```
            edifipiscip50.setCheckState(0, Qt.Unchecked)
            edifiruina50 = QTreeWidgetItem(edificacion50)
            edifiruina50.setFlags(edifiruina50.flags() |
Qt.ItemIsUserCheckable)
            edifiruina50.setText(0, "Edificació en ruïnes CV50")
            edifiruina50.setCheckState(0, Qt.Unchecked)
            edifiruinap50 = QTreeWidgetItem(edificacion50)
            edifiruinap50.setFlags(edifiruinap50.flags() |
Qt.ItemIsUserCheckable)
            edifiruinap50.setText(0, "Edificació en ruïnes puntual
CV50")
            edifiruinap50.setCheckState(0, Qt.Unchecked)
            edifextrac50 = QTreeWidgetItem(edificacion50)
            edifextrac50.setFlags(edifextrac50.flags() |
Qt.ItemIsUserCheckable)
            edifextrac50.setText(0, "Activitat extractiva CV50")
            edifextrac50.setCheckState(0, Qt.Unchecked)
            edifextracp50 = QTreeWidgetItem(edificacion50)
            edifextracp50.setFlags(edifextracp50.flags() |
Qt.ItemIsUserCheckable)
            edifextracp50.setText(0, "Activitat extractiva puntual
CV50")
            edifextracp50.setCheckState(0, Qt.Unchecked)
            edifdeposit50 = QTreeWidgetItem(edificacion50)
            edifdeposit50.setFlags(edifdeposit50.flags() |
Qt.ItemIsUserCheckable)
            edifdeposit50.setText(0, "Dipòsit")
            edifdeposit50.setCheckState(0, Qt.Unchecked)
            edifobrac50 = QTreeWidgetItem(edificacion50)
            edifobrac50.setFlags(edifobrac50.flags() |
Qt.ItemIsUserCheckable)
            edifobrac50.setText(0, "Obra de contenció")
            edifobrac50.setCheckState(0, Qt.Unchecked)
            edifinverna50 = QTreeWidgetItem(edificacion50)
            edifinverna50.setFlags(edifinverna50.flags() |
Qt.ItemIsUserCheckable)
            edifinverna50.setText(0, "Hivernacle")
            edifinverna50.setCheckState(0, Qt.Unchecked)
            edifpatio50 = QTreeWidgetItem(edificacion50)
            edifpatio50.setFlags(edifpatio50.flags() |
Qt.ItemIsUserCheckable)
            edifpatio50.setText(0, "Pati")
            edifpatio50.setCheckState(0, Qt.Unchecked)
            edifmuralla50 = QTreeWidgetItem(edificacion50)
            edifmuralla50.setFlags(edifmuralla50.flags() |
Qt.ItemIsUserCheckable)
            edifmuralla50.setText(0, "Muralla històrica")
            edifmuralla50.setCheckState(0, Qt.Unchecked)
            instalacion50 = QTreeWidgetItem(basecartografica50)
            instalacion50.setFlags(instalacion50.flags() |
Qt.ItemIsTristate | Qt.ItemIsUserCheckable)
            instalacion50.setText(0, "Instal·lacions i equipaments")
            instalacion50.setCheckState(0, Qt.Unchecked)
            instadota50 = QTreeWidgetItem(instalacion50)
            instadota50.setFlags(instadota50.flags() |
Qt.ItemIsUserCheckable)
            instadota50.setText(0, "Espai dotacional")
            instadota50.setCheckState(0, Qt.Unchecked)
            instacircu50 = QTreeWidgetItem(instalacion50)
            instacircu50.setFlags(instacircu50.flags() |
Qt.ItemIsUserCheckable)
```

```
            instacircu50.setText(0, "Circuit")
            instacircu50.setCheckState(0, Qt.Unchecked)
            instapoli50 = QTreeWidgetItem(instalacion50)
            instapoli50.setFlags(instapoli50.flags() |
Qt.ItemIsUserCheckable)
            instapoli50.setText(0, "Instal·lació de transport
poligonal")
            instapoli50.setCheckState(0, Qt.Unchecked)
            instalinea50 = QTreeWidgetItem(instalacion50)
            instalinea50.setFlags(instalinea50.flags() |
Qt.ItemIsUserCheckable)
            instalinea50.setText(0, "Instal·lació de transport
lineal")
            instalinea50.setCheckState(0, Qt.Unchecked)
            instapunt50 = QTreeWidgetItem(instalacion50)
            instapunt50.setFlags(instapunt50.flags() |
Qt.ItemIsUserCheckable)
            instapunt50.setText(0, "Instal·lació de transport
puntual")
            instapunt50.setCheckState(0, Qt.Unchecked)
            instadotpun50 = QTreeWidgetItem(instalacion50)
            instadotpun50.setFlags(instadotpun50.flags() |
Qt.ItemIsUserCheckable)
            instadotpun50.setText(0, "Espai dotacional puntual")
            instadotpun50.setCheckState(0, Qt.Unchecked)
            instaelect50 = QTreeWidgetItem(instalacion50)
            instaelect50.setFlags(instaelect50.flags() |
Qt.ItemIsUserCheckable)
            instaelect50.setText(0, "Línia elèctrica")
            instaelect50.setCheckState(0, Qt.Unchecked)
            instavertic50 = QTreeWidgetItem(instalacion50)
            instavertic50.setFlags(instavertic50.flags() |
Qt.ItemIsUserCheckable)
            instavertic50.setText(0, "Vèrtexs geodèsics")
            instavertic50.setCheckState(0, Qt.Unchecked)
            viascomu50 = QTreeWidgetItem(basecartografica50)
            viascomu50.setFlags(viascomu50.flags() | Qt.ItemIsTristate
| Qt.ItemIsUserCheckable)
            viascomu50.setText(0, "Vies de Comunicació")
            viascomu50.setCheckState(0, Qt.Unchecked)
            instared50 = QTreeWidgetItem(viascomu50)
            instared50.setFlags(instared50.flags() |
Qt.ItemIsUserCheckable)
            instared50.setText(0, "Xarxa de Comunicacions")
            instared50.setCheckState(0, Qt.Unchecked)
            instaferroc50 = QTreeWidgetItem(viascomu50)
            instaferroc50.setFlags(instaferroc50.flags() |
Qt.ItemIsUserCheckable)
            instaferroc50.setText(0, "Ferrocarril")
            instaferroc50.setCheckState(0, Qt.Unchecked)
            instaportal0 = QTreeWidgetItem(viascomu50)
            instaportal0.setFlags(instaportal0.flags() |
Qt.ItemIsUserCheckable)
            instaportal0.setText(0, "Portals/PKs")
            instaportal0.setCheckState(0, Qt.Unchecked)
            instabocat50 = QTreeWidgetItem(viascomu50)
            instabocat50.setFlags(instabocat50.flags() |
Qt.ItemIsUserCheckable)
            instabocat50.setText(0, "Boca de túnel")
            instabocat50.setCheckState(0, Qt.Unchecked)
            instaredele50 = QTreeWidgetItem(viascomu50)
```

```
            instaredele50.setFlags(instaredele50.flags() |
Qt.ItemIsUserCheckable)
            instaredele50.setText(0, "Xarxa viària elevada")
            instaredele50.setCheckState(0, Qt.Unchecked)
            instaferrel50 = QTreeWidgetItem(viascomu50)
            instaferrel50.setFlags(instaferrel50.flags() |
Qt.ItemIsUserCheckable)
            instaferrel50.setText(0, "Ferrocarril elevat")
            instaferrel50.setCheckState(0, Qt.Unchecked)
            instaviarisub50 = QTreeWidgetItem(viascomu50)
            instaviarisub50.setFlags(instaviarisub50.flags() |
Qt.ItemIsUserCheckable)
            instaviarisub50.setText(0, "Xarxa viària subterrània")
            instaviarisub50.setCheckState(0, Qt.Unchecked)
            instaviarisub50 = QTreeWidgetItem(viascomu50)
            instaviarisub50.setFlags(instaviarisub50.flags() |
Qt.ItemIsUserCheckable)
            instaviarisub50.setText(0, "Ferrocarril subterrani")
            instaviarisub50.setCheckState(0, Qt.Unchecked)
            basecartografica100 = QTreeWidgetItem(carto)
            basecartografica100.setText(0, "Base Cartogràfica CV100")
            orografia100 = QTreeWidgetItem(basecartografica100)
            orografia100.setFlags(orografia100.flags() |
Qt.ItemIsTristate | Qt.ItemIsUserCheckable)
            orografia100.setText(0, "Orografia")
            orografia100.setCheckState(0, Qt.Unchecked)
            costa100 = QTreeWidgetItem(orografia100)
            costa100.setFlags(costa100.flags() |
Qt.ItemIsUserCheckable)
            costa100.setText(0, "Línia de costa CV100")
            costa100.setCheckState(0, Qt.Unchecked)
            curvas100 = QTreeWidgetItem(orografia100)
            curvas100.setFlags(curvas100.flags() |
Qt.ItemIsUserCheckable)
            curvas100.setText(0, "Corbes de nivell CV100")
            curvas100.setCheckState(0, Qt.Unchecked)
            cota100 = QTreeWidgetItem(orografia100)
            cota100.setFlags(cota100.flags() | Qt.ItemIsUserCheckable)
            cota100.setText(0, "Punts de cota CV100")
            cota100.setCheckState(0, Qt.Unchecked)
            hidrografia100 = QTreeWidgetItem(basecartografica100)
            hidrografia100.setFlags(hidrografia100.flags() |
Qt.ItemIsTristate | Qt.ItemIsUserCheckable)
            hidrografia100.setText(0, "Hidrografia")
            hidrografia100.setCheckState(0, Qt.Unchecked)
            curson100 = QTreeWidgetItem(hidrografia100)
            curson100.setFlags(curson100.flags() |
Qt.ItemIsUserCheckable)
            curson100.setText(0, "Curs natural CV100")
            curson100.setCheckState(0, Qt.Unchecked)
            lecho100 = QTreeWidgetItem(hidrografia100)
            lecho100.setFlags(lecho100.flags() |
Qt.ItemIsUserCheckable)
            lecho100.setText(0, "Llit natural CV100")
            lecho100.setCheckState(0, Qt.Unchecked)
            lago100 = QTreeWidgetItem(hidrografia100)
            lago100.setFlags(lago100.flags() | Qt.ItemIsUserCheckable)
            lago100.setText(0, "Llac CV100")
            lago100.setCheckState(0, Qt.Unchecked)
            embalse100 = QTreeWidgetItem(hidrografia100)
            embalse100.setFlags(embalse100.flags() |
```

```
Qt.ItemIsUserCheckable)
            embalse100.setText(0, "Embassament CV100")
            embalse100.setCheckState(0, Qt.Unchecked)
            isla100 = QTreeWidgetItem(hidrografia100)
            isla100.setFlags(isla100.flags() | Qt.ItemIsUserCheckable)
            isla100.setText(0, "Illa CV100")
            isla100.setCheckState(0, Qt.Unchecked)
            edifica100 = QTreeWidgetItem(basecartografica100)
            edifica100.setFlags(edifica100.flags() | Qt.ItemIsTristate
| Qt.ItemIsUserCheckable)
            edifica100.setText(0, "Edificacions i construccions")
            edifica100.setCheckState(0, Qt.Unchecked)
            edificaed100 = QTreeWidgetItem(edifica100)
            edificaed100.setFlags(edificaed100.flags() |
Qt.ItemIsUserCheckable)
            edificaed100.setText(0, "Edificació CV100")
            edificaed100.setCheckState(0, Qt.Unchecked)
            edificaedp100 = QTreeWidgetItem(edifica100)
            edificaedp100.setFlags(edificaedp100.flags() |
Qt.ItemIsUserCheckable)
            edificaedp100.setText(0, "Edificació puntual CV100")
            edificaedp100.setCheckState(0, Qt.Unchecked)
            piscina100 = QTreeWidgetItem(edifica100)
            piscina100.setFlags(piscina100.flags() |
Qt.ItemIsUserCheckable)
            piscina100.setText(0, "Piscina/Bassa CV100")
            piscina100.setCheckState(0, Qt.Unchecked)
            piscinap100 = QTreeWidgetItem(edifica100)
            piscinap100.setFlags(piscinap100.flags() |
Qt.ItemIsUserCheckable)
            piscinap100.setText(0, "Piscina/Bassa puntual CV100")
            piscinap100.setCheckState(0, Qt.Unchecked)
            edificaedr100 = QTreeWidgetItem(edifica100)
            edificaedr100.setFlags(edificaedr100.flags() |
Qt.ItemIsUserCheckable)
            edificaedr100.setText(0, "Edificació en ruïnes CV100")
            edificaedr100.setCheckState(0, Qt.Unchecked)
            edificaedrp100 = QTreeWidgetItem(edifica100)
            edificaedrp100.setFlags(edificaedrp100.flags() |
Qt.ItemIsUserCheckable)
            edificaedrp100.setText(0, "Edificació en ruïnes puntual
CV100")
            edificaedrp100.setCheckState(0, Qt.Unchecked)
            actext100 = QTreeWidgetItem(edifica100)
            actext100.setFlags(actext100.flags() |
Qt.ItemIsUserCheckable)
            actext100.setText(0, "Activitat extractiva CV100")
            actext100.setCheckState(0, Qt.Unchecked)
            actextp100 = QTreeWidgetItem(edifica100)
            actextp100.setFlags(actextp100.flags() |
Qt.ItemIsUserCheckable)
            actextp100.setText(0, "Activitat extractiva puntual
CV100")
            actextp100.setCheckState(0, Qt.Unchecked)
            obracon100 = QTreeWidgetItem(edifica100)
            obracon100.setFlags(obracon100.flags() |
Qt.ItemIsUserCheckable)
            obracon100.setText(0, "Obra de contenció CV100")
            obracon100.setCheckState(0, Qt.Unchecked)
            inverna100 = QTreeWidgetItem(edifica100)
            inverna100.setFlags(inverna100.flags() |
```

```
Qt.ItemIsUserCheckable)
            inverna100.setText(0, "Hivernacle CV100")
            inverna100.setCheckState(0, Qt.Unchecked)
            patio100 = QTreeWidgetItem(edifica100)
            patio100.setFlags(patio100.flags() |
Qt.ItemIsUserCheckable)
            patio100.setText(0, "Pati CV100")
            patio100.setCheckState(0, Qt.Unchecked)
            muralla100 = QTreeWidgetItem(edifica100)
            muralla100.setFlags(muralla100.flags() |
Qt.ItemIsUserCheckable)
            muralla100.setText(0, "Muralla històrica CV100")
            muralla100.setCheckState(0, Qt.Unchecked)
            instala100 = QTreeWidgetItem(basecartografica100)
            instala100.setFlags(instala100.flags() | Qt.ItemIsTristate
| Qt.ItemIsUserCheckable)
            instala100.setText(0, "Instal·lacions y equipaments")
            instala100.setCheckState(0, Qt.Unchecked)
            instalinea100 = QTreeWidgetItem(instala100)
            instalinea100.setFlags(instalinea100.flags() |
Qt.ItemIsUserCheckable)
            instalinea100.setText(0, "Línia elèctrica CV100")
            instalinea100.setCheckState(0, Qt.Unchecked)
            vertices100 = QTreeWidgetItem(instala100)
            vertices100.setFlags(vertices100.flags() |
Qt.ItemIsUserCheckable)
            vertices100.setText(0, "Vèrtexs geodèsics CV100")
            vertices100.setCheckState(0, Qt.Unchecked)
            dotapunt100 = QTreeWidgetItem(instala100)
            dotapunt100.setFlags(dotapunt100.flags() |
Qt.ItemIsUserCheckable)
            dotapunt100.setText(0, "Espai dotacional puntual CV100")
            dotapunt100.setCheckState(0, Qt.Unchecked)
            transpunt100 = QTreeWidgetItem(instala100)
            transpunt100.setFlags(transpunt100.flags() |
Qt.ItemIsUserCheckable)
            transpunt100.setText(0, "Instal·lació de transport puntual
CV100")
            transpunt100.setCheckState(0, Qt.Unchecked)
            deposito100 = QTreeWidgetItem(instala100)
            deposito100.setFlags(deposito100.flags() |
Qt.ItemIsUserCheckable)
            deposito100.setText(0, "Depòsit CV100")
            deposito100.setCheckState(0, Qt.Unchecked)
            translin100 = QTreeWidgetItem(instala100)
            translin100.setFlags(translin100.flags() |
Qt.ItemIsUserCheckable)
            translin100.setText(0, "Instal·lació de transport lineal
CV100")
            translin100.setCheckState(0, Qt.Unchecked)
            transpol100 = QTreeWidgetItem(instala100)
            transpol100.setFlags(transpol100.flags() |
Qt.ItemIsUserCheckable)
            transpol100.setText(0, "Instal·lació de transport
poligonal CV100")
            transpol100.setCheckState(0, Qt.Unchecked)
            espadot100 = QTreeWidgetItem(instala100)
            espadot100.setFlags(espadot100.flags() |
Qt.ItemIsUserCheckable)
            espadot100.setText(0, "Espai dotacional CV100")
            espadot100.setCheckState(0, Qt.Unchecked)
```

```
            comunicacion100 = QTreeWidgetItem(basecartografica100)
            comunicacion100.setFlags(comunicacion100.flags() |
Qt.ItemIsTristate | Qt.ItemIsUserCheckable)
            comunicacion100.setText(0, "Vies de comunicació")
            comunicacion100.setCheckState(0, Qt.Unchecked)
            comured100 = QTreeWidgetItem(comunicacion100)
            comured100.setFlags(comured100.flags() |
Qt.ItemIsUserCheckable)
            comured100.setText(0, "Xarxa viària CV100")
            comured100.setCheckState(0, Qt.Unchecked)
            ferro100 = QTreeWidgetItem(comunicacion100)
            ferro100.setFlags(ferro100.flags() |
Qt.ItemIsUserCheckable)
            ferro100.setText(0, "Ferrocarril CV100")
            ferro100.setCheckState(0, Qt.Unchecked)
            tunel100 = QTreeWidgetItem(comunicacion100)
            tunel100.setFlags(tunel100.flags() |
Qt.ItemIsUserCheckable)
            tunel100.setText(0, "Boca de túnel CV100")
            tunel100.setCheckState(0, Qt.Unchecked)
            comurede100 = QTreeWidgetItem(comunicacion100)
            comurede100.setFlags(comurede100.flags() |
Qt.ItemIsUserCheckable)
            comurede100.setText(0, "Xarxa viària elevada CV100")
            comurede100.setCheckState(0, Qt.Unchecked)
            ferroe100 = QTreeWidgetItem(comunicacion100)
            ferroe100.setFlags(ferroe100.flags() |
Qt.ItemIsUserCheckable)
            ferroe100.setText(0, "Ferrocarril elevat CV100")
            ferroe100.setCheckState(0, Qt.Unchecked)
            comureds100 = QTreeWidgetItem(comunicacion100)
            comureds100.setFlags(comureds100.flags() |
Qt.ItemIsUserCheckable)
            comureds100.setText(0, "Xarxa viària subterrània CV100")
            comureds100.setCheckState(0, Qt.Unchecked)
            ferros100 = QTreeWidgetItem(comunicacion100)
            ferros100.setFlags(ferros100.flags() |
Qt.ItemIsUserCheckable)
            ferros100.setText(0, "Ferrocarril subterrani CV100")
            ferros100.setCheckState(0, Qt.Unchecked)
            fechasbases = QTreeWidgetItem(carto)
            fechasbases.setText(0, "Dates Bases cartogràfiques")
            fechasvuelos = QTreeWidgetItem(fechasbases)
            fechasvuelos.setFlags(fechasvuelos.flags() |
Qt.ItemIsUserCheckable)
            fechasvuelos.setText(0, "Dates de vol")
            fechasvuelos.setCheckState(0, Qt.Unchecked)
            redtransp = QTreeWidgetItem(carto)
            redtransp.setText(0, "Xarxa de Transports")
            redcarretera = QTreeWidgetItem(redtransp)
            redcarretera.setFlags(redcarretera.flags() |
Qt.ItemIsTristate | Qt.ItemIsUserCheckable)
            redcarretera.setText(0, "Transport Carretera")
            redcarretera.setCheckState(0, Qt.Unchecked)
            sistviari = QTreeWidgetItem(redcarretera)
            sistviari.setFlags(sistviari.flags() | Qt.ItemIsTristate |
Qt.ItemIsUserCheckable)
            sistviari.setText(0, "Sistema Viari")
            sistviari.setCheckState(0, Qt.Unchecked)
            cataviario = QTreeWidgetItem(sistviari)
            cataviario.setFlags(cataviario.flags() | Qt.ItemIsTristate
```

```
| Qt.ItemIsUserCheckable)
            cataviario.setText(0, "Catàleg Viari")
            cataviario.setCheckState(0, Qt.Unchecked)
            cataviario1 = QTreeWidgetItem(cataviario)
            cataviario1.setFlags(cataviario1.flags() |
Qt.ItemIsUserCheckable)
            cataviario1.setText(0, "Catàleg viari escala 1")
            cataviario1.setCheckState(0, Qt.Unchecked)
            cataviario2 = QTreeWidgetItem(cataviario)
            cataviario2.setFlags(cataviario1.flags() |
Qt.ItemIsUserCheckable)
            cataviario2.setText(0, "Catàleg viari escala 2")
            cataviario2.setCheckState(0, Qt.Unchecked)
            cataviario3 = QTreeWidgetItem(cataviario)
            cataviario3.setFlags(cataviario3.flags() |
Qt.ItemIsUserCheckable)
            cataviario3.setText(0, "Catàleg viari escala 3")
            cataviario3.setCheckState(0, Qt.Unchecked)
            cataviario4 = QTreeWidgetItem(cataviario)
            cataviario4.setFlags(cataviario4.flags() |
Qt.ItemIsUserCheckable)
            cataviario4.setText(0, "Catàleg viari escala 4")
            cataviario4.setCheckState(0, Qt.Unchecked)
            puntokil = QTreeWidgetItem(sistviari)
            puntokil.setFlags(puntokil.flags() |
Qt.ItemIsUserCheckable)
            puntokil.setText(0, "Punts quilomètrics")
            puntokil.setCheckState(0, Qt.Unchecked)
            caminos = QTreeWidgetItem(sistviari)
            caminos.setFlags(puntokil.flags() |
Qt.ItemIsUserCheckable)
            caminos.setText(0, "Camins")
            caminos.setCheckState(0, Qt.Unchecked)
            urbana = QTreeWidgetItem(sistviari)
            urbana.setFlags(urbana.flags() | Qt.ItemIsTristate |
Qt.ItemIsUserCheckable)
            urbana.setText(0, "Urbana")
            urbana.setCheckState(0, Qt.Unchecked)
            portal = QTreeWidgetItem(urbana)
            portal.setFlags(portal.flags() | Qt.ItemIsUserCheckable)
            portal.setText(0, "Portal")
            portal.setCheckState(0, Qt.Unchecked)
            listacalles = QTreeWidgetItem(urbana)
            listacalles.setFlags(listacalles.flags() |
Qt.ItemIsUserCheckable)
            listacalles.setText(0, "Llistat de carrers")
            listacalles.setCheckState(0, Qt.Unchecked)
            infraestruc = QTreeWidgetItem(redcarretera)
            infraestruc.setFlags(infraestruc.flags() |
Qt.ItemIsTristate | Qt.ItemIsUserCheckable)
            infraestruc.setText(0, "Infraestructures e
instal·lacions")
            infraestruc.setCheckState(0, Qt.Unchecked)
            estautob = QTreeWidgetItem(infraestruc)
            estautob.setFlags(estautob.flags() |
Qt.ItemIsUserCheckable)
            estautob.setText(0, "Estació d'autobusos")
            estautob.setCheckState(0, Qt.Unchecked)
            peaje = QTreeWidgetItem(infraestruc)
            peaje.setFlags(peaje.flags() | Qt.ItemIsUserCheckable)
            peaje.setText(0, "Peatge")
```

```
            peaje.setCheckState(0, Qt.Unchecked)
            areaserv = QTreeWidgetItem(infraestruc)
            areaserv.setFlags(areaserv.flags() |
Qt.ItemIsUserCheckable)
            areaserv.setText(0, "Àrees de servei")
            areaserv.setCheckState(0, Qt.Unchecked)
            areaservin = QTreeWidgetItem(infraestruc)
            areaservin.setFlags(areaservin.flags() |
Qt.ItemIsUserCheckable)
            areaservin.setText(0, "Aparcament de servicis viaris
hivernal")
            areaservin.setCheckState(0, Qt.Unchecked)
            puente = QTreeWidgetItem(infraestruc)
            puente.setFlags(puente.flags() | Qt.ItemIsUserCheckable)
            puente.setText(0, "Pont")
            puente.setCheckState(0, Qt.Unchecked)


        self.tree_toc.expandToDepth(1)  # Tree level in which it is
presented when loaded

    """ Layer search function in the QtreeWidget """

    def searchLayer(self):
        if self.txt_busca_capa.text() == 'Busca capa...': # Help text
            pass
        else:  # Start the search if the text is not the help text
            text_capa = self.txt_busca_capa.text()  # Assign the
search text to the variable
            self.list_capa.clear()
            if len(text_capa) > 2:  # Star the search if the text
lenght is bigger than 3 characters
                self.list_capa.show()
                if self.rdb_cas.isChecked() == True: # Search in
spanish
                    i = 0
                    for item in capasIDEV:  # We go through all the
layers
                            if text_capa.lower() in
capasIDEV[i]['nombre_cas'].lower():

self.list_capa.addItem(capasIDEV[i]['nombre_cas']) # If a layer
matches the search pattern it is added to the listWisget
                            i = i + 1
                if self.rdb_val.isChecked() == True:  # Search in
valencian
                    i = 0
                    for item in capasIDEV: # We go through all the
layers
                        if text_capa.lower() in
capasIDEV[i]['nombre_val'].lower():

self.list_capa.addItem(capasIDEV[i]['nombre_val']) # If a layer
matches the search pattern it is added to the listWisget
                        i = i + 1
            else:
                self.list_capa.hide() # Hide the listWidget of the
results of the layers search

    """ Function that when selecting a layer of the QTreeWidget
activates the link button with the catalog """
```

```
    #@QtCore.pyqtSlot(QtWidgets.QTreeWidgetItem, int)
    def selectTreeLayer(self):
        self.tree_toc.clearSelection() # Clear the actual layers
selection
        nom_capa = self.list_capa.currentItem().text() # We assign to
the variable the selected layer of the QTreeWidget
        for item in self.tree_toc.findItems(nom_capa, Qt.MatchContains
| Qt.MatchRecursive):
            self.tree_toc.setCurrentItem(item) # Capture of the
selected item by clicking
            item.setSelected(True) # Select the item in the
QTreeWidget
            item.setExpanded(True) # Expand the QtreeWidget at the
item level
        if self.rdb_cas.isChecked() == True:
            self.txt_busca_capa.setText("Busca capa...")
        if self.rdb_val.isChecked() == True:
            self.txt_busca_capa.setText("Busca capa...")
        self.list_capa.hide()

    """ Function that closes the plugin """

    def closeEvent(self, event):
        self.closingPlugin.emit()
        event.accept()

""" We define the dictionary of layers outside the functions so that
it is accessible in all """

path_capas_json = os.path.dirname(os.path.realpath(__file__)) +
'/params.json' # Path of the JSON file with the definition of each
layer
with open(path_capas_json, "r") as read_file:  # Read the JSON file
    capasIDEV = json.load(read_file)  # Store in a dictionary the JSON
file, each element is a layer


""" We define the layer data structure as a class """

class capaIDEV:
    def __init__(self, nombre_cas, nombre_val, aviso_cas, aviso_val,
metadato, url):
        self.nombre = nombre_cas
        self.tipo = nombre_val
        self.aviso = aviso_cas
        self.aviso = aviso_val
        self.metadato = metadato
        self.url = url

    def __str__(self):
        cadena = 'Nombre castellano: \n'.format(self.nombre_cas)
        cadena = 'Nombre valenciano: \n'.format(self.nombre_val)
        cadena = cadena + 'Aviso castellano:
\n'.format(self.aviso_cas)
        cadena = cadena + 'Aviso valenciano:
\n'.format(self.aviso_val)
        cadena = cadena + 'Metadato UIDD: \n'.format(self.metadato)
        cadena = cadena + 'URL conexión: \n'.format(self.url)
        return cadena
```

Código del fichero "*idev_visor_dockwidget_base.ui*".

```
# -*- coding: utf-8 -*-
"""
/***************************************************************************
 Idev
                                 A QGIS plugin
 Infraestructura Valenciana de Dades Espacials
 Generated by Plugin Builder: http://g-sherman.github.io/Qgis-Plugin-
Builder/
                             -------------------
        begin                : 2020-08-13
        git sha              : $Format:%H$
        copyright            : (C) 2020 by Alfonso Moya Fuero
        email                : moya_alf@gva.es

 ***************************************************************************
 *****/

/***************************************************************************
 *
 *
 *   This program is free software; you can redistribute it and/or
modify  *
 *   it under the terms of the GNU General Public License as published
by  *
 *   the Free Software Foundation; either version 2 of the License, or
*
 *   (at your option) any later version.
*
 *
*

 ***************************************************************************
 *****/
"""
from qgis.PyQt.QtCore import QSettings, QTranslator, QCoreApplication,
Qt
from qgis.PyQt.QtGui import QIcon
from qgis.PyQt.QtWidgets import QAction
# Initialize Qt resources from file resources.py
from .resources import *

# Import the code for the DockWidget
from .idev_visor_dockwidget import IdevDockWidget
import os.path

class Idev:
    """QGIS Plugin Implementation."""

    def __init__(self, iface):
        """Constructor.

        :param iface: An interface instance that will be passed to
this class
            which provides the hook by which you can manipulate the
QGIS
            application at run time.
        :type iface: QgsInterface
```

```
        """
        # Save reference to the QGIS interface
        self.iface = iface

        # initialize plugin directory
        self.plugin_dir = os.path.dirname(__file__)

        # initialize locale
        locale = QSettings().value('locale/userLocale')[0:2]
        locale_path = os.path.join(
            self.plugin_dir,
            'i18n',
            'Idev_{}.qm'.format(locale))

        if os.path.exists(locale_path):
            self.translator = QTranslator()
            self.translator.load(locale_path)
            QCoreApplication.installTranslator(self.translator)

        # Declare instance attributes
        self.actions = []
        self.menu = self.tr(u'&IDEV')
        # TODO: We are going to let the user set this up in a future
iteration
        self.toolbar = self.iface.addToolBar(u'Idev')
        self.toolbar.setObjectName(u'Idev')

        #print "** INITIALIZING Idev"

        self.pluginIsActive = False
        self.dockwidget = None


    # noinspection PyMethodMayBeStatic
    def tr(self, message):
        """Get the translation for a string using Qt translation API.

        We implement this ourselves since we do not inherit QObject.

        :param message: String for translation.
        :type message: str, QString

        :returns: Translated version of message.
        :rtype: QString
        """
        # noinspection PyTypeChecker,PyArgumentList,PyCallByClass
        return QCoreApplication.translate('Idev', message)


    def add_action(
        self,
        icon_path,
        text,
        callback,
        enabled_flag=True,
        add_to_menu=True,
        add_to_toolbar=True,
        status_tip=None,
        whats_this=None,
        parent=None):
        """Add a toolbar icon to the toolbar.
```

```
        :param icon_path: Path to the icon for this action. Can be a
resource
            path (e.g. ':/plugins/foo/bar.png') or a normal file
system path.
        :type icon_path: str

        :param text: Text that should be shown in menu items for this
action.
        :type text: str

        :param callback: Function to be called when the action is
triggered.
        :type callback: function

        :param enabled_flag: A flag indicating if the action should be
enabled
            by default. Defaults to True.
        :type enabled_flag: bool

        :param add_to_menu: Flag indicating whether the action should
also
            be added to the menu. Defaults to True.
        :type add_to_menu: bool

        :param add_to_toolbar: Flag indicating whether the action
should also
            be added to the toolbar. Defaults to True.
        :type add_to_toolbar: bool

        :param status_tip: Optional text to show in a popup when mouse
pointer
            hovers over the action.
        :type status_tip: str

        :param parent: Parent widget for the new action. Defaults
None.
        :type parent: QWidget

        :param whats_this: Optional text to show in the status bar
when the
            mouse pointer hovers over the action.

        :returns: The action that was created. Note that the action is
also
            added to self.actions list.
        :rtype: QAction
        """

        icon = QIcon(icon_path)
        action = QAction(icon, text, parent)
        action.triggered.connect(callback)
        action.setEnabled(enabled_flag)

        if status_tip is not None:
            action.setStatusTip(status_tip)

        if whats_this is not None:
            action.setWhatsThis(whats_this)

        if add_to_toolbar:
```

```
            self.toolbar.addAction(action)

        if add_to_menu:
            self.iface.addPluginToMenu(
                self.menu,
                action)

        self.actions.append(action)

        return action


    def initGui(self):
        """Create the menu entries and toolbar icons inside the QGIS
GUI."""

        icon_path = ':/plugins/idev_visor/icon.png'
        self.add_action(
            icon_path,
            text=self.tr(u'IDEV'),
            callback=self.run,
            parent=self.iface.mainWindow())

    #--------------------------------------------------------------------
---------

    def onClosePlugin(self):
        """Cleanup necessary items here when plugin dockwidget is
closed"""

        #print "** CLOSING Idev"

        # disconnects
        self.dockwidget.closingPlugin.disconnect(self.onClosePlugin)

        # remove this statement if dockwidget is to remain
        # for reuse if plugin is reopened
        # Commented next statement since it causes QGIS crashe
        # when closing the docked window:
        # self.dockwidget = None

        self.pluginIsActive = False


    def unload(self):
        """Removes the plugin menu item and icon from QGIS GUI."""

        #print "** UNLOAD Idev"

        for action in self.actions:
            self.iface.removePluginMenu(
                self.tr(u'&IDEV'),
                action)
            self.iface.removeToolBarIcon(action)
        # remove the toolbar
        del self.toolbar

    #--------------------------------------------------------------------
---------

    def run(self):
```

```
        """Run method that loads and starts the plugin"""

        if not self.pluginIsActive:
            self.pluginIsActive = True

            #print "** STARTING Idev"

            # dockwidget may not exist if:
            #    first run of plugin
            #    removed on close (see self.onClosePlugin method)
            if self.dockwidget == None:
                # Create the dockwidget (after translation) and keep
reference
                self.dockwidget = IdevDockWidget()

            # connect to provide cleanup on closing of dockwidget
            self.dockwidget.closingPlugin.connect(self.onClosePlugin)

            # show the dockwidget
            # TODO: fix to allow choice of dock location
            self.iface.addDockWidget(Qt.RightDockWidgetArea,
self.dockwidget)
            self.dockwidget.show()
```

Diccionario de datos, fichero "*params.json*".

Se adjunta una muestra del fichero, no es necesario ajuntar todo ya que lo importante es como se han definido los campos en el fichero "params.json". Si se quiere consultar el contenido completo, este fichero está en el directorio de instalación del complemento. Este directorio se puede consultar a partir del menú de *QGIS "Configuración" / "Perfiles de usuario" / "Abrir la carpeta del perfil activo"* y se encuentra en ese directorio, en la carpeta *"Python" / "plugins" / "idev_visor"*

```
[
  {
    "nombre_cas": "Nomenclator Toponimico Valenciano",
    "nombre_val": "Nomenclàtor Toponímic Valencià",
    "aviso_cas": "El Nomenclátor es una base de datos espacial que
contiene más de 120.000 topónimos que hacen referencia a la toponimia
mayor (poblaciones, ríos principales, sierras principales, etc) y
toponimia menor (ríos secundarios, barrancos, montañas, caminos,
sendas, parajes, cuevas, fuentes, entidades menores de población,
etc.) de la Comunitat Valenciana. El Nomenclátor es un proyecto que se
está actualizando constantemente a través del trabajo conjunto entre
técnicos, lingüistas y colaboradores del Institut Cartogràfic Valencià
(http://www.icv.gva.es/ca) y de la Acadèmia Valenciana de la Llengua
(http://www.avl.gva.es/web/avl/info-castellano).",
    "aviso_val": "El Nomenclàtor és una base de dades espacials que
conté més de 120.000 topònims que fan referència a la toponímia major
(poblacions, rius principals, serres principals, etc) i toponímia
menor (rius secundaris, barrancs, muntanyes, camins, sendes, paratges,
coves, fonts, entitats menors de població; etc.) de la Comunitat
Valenciana. El Nomenclàtor és un projecte que s'està actualitzant
constantment a través del treball conjunt entre tècnics, lingüistes i
col·laboradors de l'Institut Cartogràfic Valencià
(http://www.icv.gva.es) i de l'Acadèmia Valenciana de la Llengua
(http://www.avl.gva.es/).",
    "metadato": "spaicvNomenclatorCV_ICV",
    "url":
```

```
"crs=EPSG:25830&dpiMode=7&format=image/png&layers=NOMENCLATOR_ICV&styl
es=default&url=https://terramapas.icv.gva.es/toponimia_base"
  },
  {
    "nombre_cas": "Puntos de cota CV05",
    "nombre_val": "Punts de cota CV05",
    "aviso_cas": "",
    "aviso_val": "",
    "metadato": "spaicvBcv05Serie2015",
    "url":
"crs=EPSG:25830&dpiMode=7&format=image/png&layers=OrogpuntosCV05&style
s&url=http://terramapas.icv.gva.es/mapa_topografico_base"
  },
  {
    "nombre_cas": "Orografía líneas CV05",
    "nombre_val": "Orografia línies CV05",
    "aviso_cas": "",
    "aviso_val": "",
    "metadato": "spaicvBcv05Serie2015",
    "url":
"crs=EPSG:25830&dpiMode=7&format=image/png&layers=OroglineasCV05&style
s&url=http://terramapas.icv.gva.es/mapa_topografico_base"
  },
.....
```